

# ArtiSynth User Interface Guide

---

**John Lloyd**

Last update: February, 2023

## Contents

<b>1</b>	<b>Introduction</b>	<b>3</b>
1.1	User configuration folder . . . . .	3
<b>2</b>	<b>Loading, Simulating and Saving Models</b>	<b>3</b>
2.1	Loading from the model menu . . . . .	3
2.2	Loading directly by class . . . . .	4
2.3	Loading from a file . . . . .	5
2.4	Loading recent models . . . . .	5
2.5	Setting a startup model . . . . .	5
2.5.1	Specifying models from the command line . . . . .	6
2.6	Simulating a model . . . . .	6
2.7	Other toolbar controls . . . . .	6
2.8	Saving a model . . . . .	7
2.9	Setting the external classpath . . . . .	8
2.10	The ArtiSynth working folder . . . . .	8
<b>3</b>	<b>The Viewer</b>	<b>9</b>
3.1	Viewer Toolbar . . . . .	9
3.2	Viewpoint control . . . . .	9
3.3	Adding additional viewers . . . . .	10
3.4	World coordinate axes . . . . .	10
3.5	Orthographic vs. perspective projection . . . . .	10
3.6	Viewer grid . . . . .	11
3.6.1	Grid units . . . . .	11
3.6.2	Axis labeling . . . . .	11
3.6.3	Grid properties . . . . .	11
3.7	Clipping planes . . . . .	13
3.7.1	Adding and removing . . . . .	14
3.7.2	Moving . . . . .	14
3.7.3	Offsets . . . . .	14
3.7.4	Enabling/disabling . . . . .	14
3.7.5	Slicing mode . . . . .	14
3.7.6	Other features . . . . .	15
3.8	Indicating 3D positions with the mouse . . . . .	15
3.9	Viewer properties . . . . .	15
3.9.1	Viewer-specific properties . . . . .	17
3.10	Mouse Bindings . . . . .	17
3.11	Keyboard shortcuts . . . . .	19

---

<b>4</b>	<b>Component Navigation and Selection</b>	<b>19</b>
4.1	The component hierarchy . . . . .	19
4.1.1	Component names and numbers . . . . .	20
4.1.2	Component path names . . . . .	20
4.2	Navigation panel selection . . . . .	21
4.2.1	Large numbers of nameless components . . . . .	21
4.3	Viewer selection . . . . .	21
4.3.1	Click and box selection . . . . .	22
4.3.2	Elliptic selection . . . . .	22
4.3.3	Selection filtering . . . . .	22
4.4	Selection display . . . . .	23
4.5	Selecting parent and ancestor components . . . . .	23
4.6	Highlighting selected components . . . . .	23
4.7	Selecting FEM nodes . . . . .	24
<b>5</b>	<b>Model Manipulation</b>	<b>27</b>
5.1	Dragger fixtures . . . . .	27
5.2	Transformer tools . . . . .	28
5.2.1	Constrained transformation . . . . .	28
5.2.2	Transformer repositioning . . . . .	29
5.2.3	Changing the transformer base frame . . . . .	29
5.2.4	Flipping transformer axes forward . . . . .	29
5.2.5	Resizing transformers . . . . .	30
5.3	Pull manipulation . . . . .	30
5.4	Marker tool . . . . .	30
<b>6</b>	<b>Editing Properties</b>	<b>31</b>
6.1	Property panels . . . . .	31
6.1.1	Inheritable properties . . . . .	32
6.2	Render properties . . . . .	32
6.2.1	Render property settings . . . . .	33
<b>7</b>	<b>The Timeline</b>	<b>34</b>
7.1	Probes and waypoints . . . . .	35
7.2	Basic timeline structure . . . . .	35
7.2.1	Play controls . . . . .	36
7.2.2	Tracks . . . . .	36
7.3	Viewing and setting waypoints . . . . .	36
7.3.1	Waypoints . . . . .	36
7.3.2	Breakpoints . . . . .	37
7.3.3	Saving and loading . . . . .	37
7.4	Tracks and probes . . . . .	37

---

7.4.1	Creating, moving, and deleting tracks . . . . .	38
7.4.2	Muting tracks . . . . .	38
7.4.3	Expanding tracks . . . . .	38
7.4.4	Grouping tracks . . . . .	38
7.5	Numeric probe displays . . . . .	38
7.5.1	Setting the range and display properties . . . . .	38
7.5.2	Large displays . . . . .	39
7.5.3	Cloning displays and exporting plots . . . . .	40
7.5.4	Legends and visibility control . . . . .	40
7.5.5	Editing and scaling data . . . . .	44
7.5.6	Smoothing data . . . . .	44
7.5.7	Interpolation control . . . . .	45
<b>8</b>	<b>Saving and Loading Probes</b>	<b>45</b>
8.1	Saving and loading probe data . . . . .	45
8.2	Exporting numeric probe data . . . . .	46
8.3	Saving and loading all probes . . . . .	47
<b>9</b>	<b>Adding and Editing Numeric Probes</b>	<b>47</b>
9.1	Adding output probes . . . . .	47
9.1.1	Creating a simple probe . . . . .	48
9.1.2	General output probes . . . . .	48
9.1.3	Using the probe editor . . . . .	49
9.2	Adding input probes . . . . .	49
9.2.1	Creating a simple probe . . . . .	49
9.2.2	General input probes . . . . .	50
9.2.3	Using the probe editor . . . . .	51
9.3	Setting probe properties . . . . .	51
<b>10</b>	<b>Point Tracing</b>	<b>52</b>
<b>11</b>	<b>Settings and Preferences</b>	<b>53</b>
11.1	Settings . . . . .	53
11.1.1	Interaction . . . . .	53
11.1.2	Simulation . . . . .	54
11.2	Preferences . . . . .	55
11.3	Layout preferences . . . . .	56
<b>12</b>	<b>Jython Interaction and Scripting</b>	<b>57</b>
12.1	Querying ArtiSynth structures and models . . . . .	58
12.2	Object creation and importing classes . . . . .	59
12.3	Running simulations and scripting . . . . .	59
12.4	Using the script menu . . . . .	60
12.5	Selecting a script file . . . . .	61
12.6	Specifying scripts on the command line . . . . .	61
12.7	Built-in functions . . . . .	62

---

<b>13 Customizing the Model and Script Menus</b>	<b>64</b>
13.1 Model menu editor . . . . .	64
13.2 Script menu editor . . . . .	66
13.3 Menu entry types . . . . .	67
13.3.1 Model . . . . .	67
13.3.2 Package . . . . .	67
13.3.3 Demo file . . . . .	68
13.3.4 Script . . . . .	68
13.3.5 Script folder . . . . .	69
13.3.6 Submenu . . . . .	69
13.3.7 Label . . . . .	70
13.3.8 Separator . . . . .	71
13.4 Command line options . . . . .	71
13.5 Demo file text format . . . . .	71
13.6 XML Menu Format . . . . .	71
13.6.1 The root elements . . . . .	72
13.6.2 Model element . . . . .	72
13.6.3 Package element . . . . .	73
13.6.4 DemoFile element . . . . .	73
13.6.5 Script element . . . . .	73
13.6.6 ScriptFolder element . . . . .	74
13.6.7 Submenu element . . . . .	74
13.6.8 Label element . . . . .	74
13.6.9 Separator element . . . . .	75
13.6.10 Hiding elements . . . . .	75
<b>14 Making Movies</b>	<b>75</b>
14.1 Recorder tab . . . . .	76
14.1.1 Region to capture . . . . .	76
14.1.2 Record options . . . . .	77
14.1.3 Other options . . . . .	78
14.2 Encoder tab . . . . .	78
14.2.1 Encoder options . . . . .	78
14.2.2 Customizing the encoder command . . . . .	79
14.3 Output size options . . . . .	79
14.4 Advanced tab . . . . .	80
14.5 Saving movie preferences . . . . .	80
14.6 Installing FFmpeg . . . . .	81

---

<b>15 Control Panels</b>	<b>81</b>
15.1 Creating control panels . . . . .	81
15.1.1 Composite property widgets . . . . .	82
15.1.2 Widgets for subproperties . . . . .	83
15.2 Editing control panels . . . . .	84
15.3 Live updating . . . . .	84
<b>16 Component Editing</b>	<b>84</b>
16.1 Generic edit operations . . . . .	84
16.1.1 Deletion . . . . .	84
16.1.2 Duplication . . . . .	85
16.1.3 Undo . . . . .	85
16.2 Editing panels . . . . .	85
16.3 Specifying position, orientation, and scaling . . . . .	85
16.4 Editing MechModels . . . . .	86
16.4.1 Adding finite element models . . . . .	86
16.4.2 Adding rigid bodies . . . . .	88
16.4.3 Adding frame markers . . . . .	88
16.4.4 Adding particles . . . . .	89
16.4.5 Adding axial springs and muscles . . . . .	90
16.4.6 Adding rigid body connectors . . . . .	91
16.4.7 Attaching particles to particles . . . . .	92
16.4.8 Attaching particles to rigid bodies . . . . .	92
16.4.9 Collision handling . . . . .	93
16.5 Editing rigid bodies . . . . .	95
16.5.1 Geometry and inertia . . . . .	96
16.6 Editing FEM models . . . . .	97
16.6.1 Adding FEM markers . . . . .	98
16.6.2 Adding muscle bundles . . . . .	98
16.7 Editing muscle bundles . . . . .	99
16.7.1 Adding fibres . . . . .	99
16.7.2 Adding element references . . . . .	99
16.7.3 Automatically setting elements and directions . . . . .	100
16.7.4 Removing fibres and element references . . . . .	101
16.8 Editing muscle exciters . . . . .	101
16.9 Editing root models . . . . .	102

## 1 Introduction

This manual describes the ArtiSynth user interface (UI), and how it can be used to edit models and interactively monitor and control their simulation.

### 1.1 User configuration folder

By default, ArtiSynth tries to store user-specific settings and configuration information in various files located under the *user configuration folder*, the path to which is given by

```
<HOME>/ArtiSynthConfig
```

where `<HOME>` is the user's home folder. Default instances of these files will be created when they are first accessed or recreated if they are later found to be corrupt or missing.

## 2 Loading, Simulating and Saving Models

The first thing an ArtiSynth user is likely to want is to load a demonstration model, and explore and simulate it.

An ArtiSynth model is defined by a Java class which is a subclass of the ArtiSynth `RootModel` component. This class builds the model, serves as the root container for all its components, and implements the `advance()` method which allows the model to be simulated.

A number of predefined demonstration models come bundled with the ArtiSynth distribution and are declared within subpackages of `artisynth.demos`. These are generally simple models that illustrate particular simulation capabilities. More complex anatomical models, including those used in various research projects and mostly focused on head and neck anatomy, are available in the separate project `artisynth_models`, which must be downloaded separately (see [www.artisynth.org/models](http://www.artisynth.org/models) for instructions). These anatomical models are declared within subpackages of `artisynth.models`.

To run user-defined models defined within Java packages that are *outside* of the project `artisynth_core`, it is necessary to arrange for their classpaths to be visible to the ArtiSynth application. One way to do this is to add the classpaths to the ArtiSynth external classpath, as described in Section 2.9. If one is using an integrated development environment (IDE) for Java compilation and execution, this linking can also be done within the IDE. More detailed information on this topic is given in the section “Making external models visible to ArtiSynth” of the ArtiSynth Installation Guide.

### 2.1 Loading from the model menu

Some models can be loaded directly using the Models menu located in the ArtiSynth menu bar (Figure 1). By default, the upper part of this menu contains a number of submenus:

- Demos - all models listed in the file `demoModels.txt` (described below);
- All Demos - every model found in `artisynth.demos` or its subpackages, arranged hierarchically.

In addition, if `artisynth_models` has also been installed, or if ArtiSynth otherwise detects the presence of the superpackage `artisynth.models`, then the Models menu will also contain:

- Models - all models listed in the file `mainModels.txt` (described below);
- All Models - every model found in `artisynth.models` or its subpackages, arranged hierarchically.

Each submenu expands out to identify a set of models. Selecting one of the models will cause it to be loaded into ArtiSynth and displayed in the viewer. Hovering over one of the entries will display the full classname of the associated `RootModel`. The files `demoModels.txt` and `mainModels.txt` are located in the `settings` subfolder of the user configuration folder (Section 1.1); default instances of these are created the first time ArtiSynth is run, and they can later be modified by the user.

The lower part of the model menu, beneath the separator, contains entries for reloading recent models (Section 2.4), loading a model from an explicitly specified class (Section 2.2), and customizing the upper part of the the model menu (Section 13).



Figure 1: The ArtiSynth model selection menu.

Model menu customization is needed to create menu entries for root models not defined within (or beneath) the packages `artisynth.demos` or `artisynth.models`.

## 2.2 Loading directly by class

As mentioned above, models are defined by subclasses of `RootModel`. A model may therefore be loaded into ArtiSynth by directly specifying the class that defines its `RootModel`. To do this, choose “Load from class ...” from the lower part of the model menu, which will bring up a model selection dialog as shown in Figure 2.

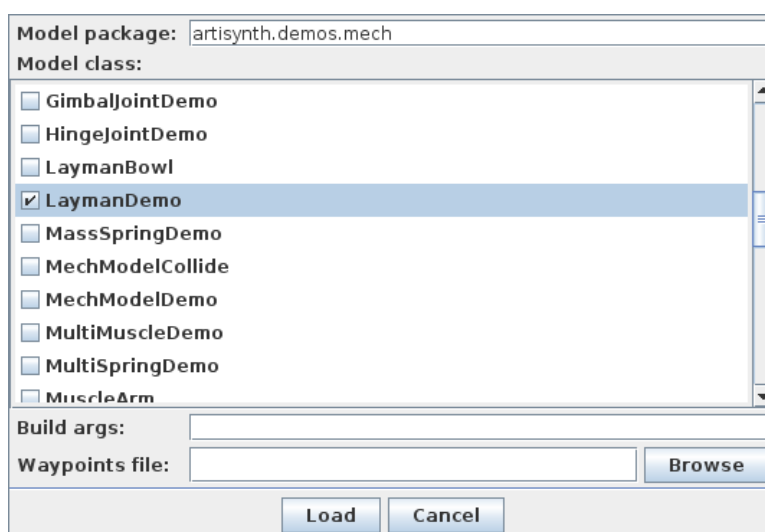


Figure 2: Dialog for selecting a model class.

Users should specify the name of the package containing the model in the “Model package” field at the top. Using the

<TAB> character in this field will invoke auto-completion based on currently known packages, while repeated use of either <TAB> or the up/down arrows will scroll through known packages. Package entry is completed using the <ENTER> key, and any models that appear in that package (but not its subpackages) will then be displayed in the “Model class” panel below it. The user can then select the desired model by clicking on it. In Figure 2, the model defined by the class `artisynth.demos.mech.LaymanDemo` has been selected.

If the model requires command-line style arguments to its `build()` method (as described in the section “Implementing the `build()` method” of the [ArtiSynth Modeling Guide](#)), these can be entered in the “Build args” field near the dialog bottom. Arguments should be separated by white space, with those containing white space placed between double quotes ‘”’. The last field, “Waypoints file”, can optionally be used to specify a file containing simulation waypoints (Section 7.3.1). As with all externally loaded waypoint data, the waypoints must match the current model structure.

When all desired settings have been made, the model can be loaded by clicking the Load button.

## 2.3 Loading from a file

Finally, it is possible to load a model from a file. Selecting “Load model ...” from the File menu will bring up a File browser that lets you select and load a model from an ArtiSynth model file. ArtiSynth model files are text-based documents that contain a hierarchical description of all the model’s components, and are typically identified by the extension `.art`.

When loading a model from a `.art` file, it is necessary to have all classes associated with that model in the current Java classpath. This can be an issue when loading files generated by other users using application-specific Java code. Two possible solutions to this are: (a) bundling the application-specific code into a `.jar` file and adding it to the external classpath (Section 2.9), or (b) making sure that the file was saved using only `artisynth_core` components, as described in Section 2.8.

## 2.4 Loading recent models

After a model has been loaded by any of the methods described above, it can be reloaded by selecting “Reload model” from the lower part of the model menu. Models which have been recently loaded can be reloaded by selecting “Load recent” from the Models menu.

## 2.5 Setting a startup model

When working repeatedly with a specific model, it can be useful to set that model to automatically load when ArtiSynth starts up. This can be done by setting the *startup model*, by choosing “Startup model ...” from the Settings menu. This will open a startup model dialog as shown in Figure 3.

The model to load can be specified either by class or by file. To specify the model by class, one uses the “Model package”, “Model class” and “Build args” fields to choose the model class and optional `build()` method arguments in the same manner as described in Section 2.2. To specify the model by file, one instead uses the “Model file” field to select a model file, as described in Section 2.3.

For either kind of model, it is also possible to use the “Waypoints file” field to specify a file of saved waypoints to be loaded along with the model. As with all externally loaded waypoint data, the waypoints must match the current model structure. Waypoints are described in Section 7.3.1.

- To save the startup model, click the Save button at the bottom of the dialog.
- To clear the startup model (so that no model is loaded), click the Clear button followed by the Save button.
- To load a specified model immediately, click the Load Model button.

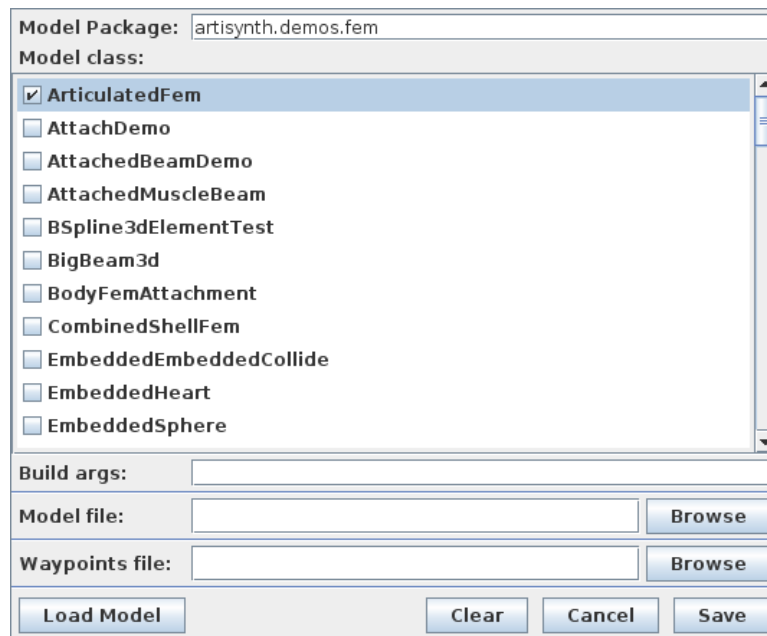


Figure 3: Dialog to set the startup model.

### 2.5.1 Specifying models from the command line

As an alternative to setting the startup model, one may instead use the `-model <classOrFileName>` command line option to specify a model to load when ArtiSynth starts up. This can be useful when running ArtiSynth from a script. The `<classOrFileName>` argument may be either a class name or a `.art` file name. If a class name is specified and `build()` method arguments are also required, these may be listed within square brackets (`[ ]`) separated by white space. For example, to load the model class `projects.MyModel` and pass it the `build()` arguments “`-size 50`”, one can invoke ArtiSynth from the command line using

```
> artisynt -model projects.MyModel [ -size 50 ]
```

Models specified from the command line override the specified startup model. To ensure that *no* model is loaded, one may specify

```
> artisynt -model none
```

## 2.6 Simulating a model

Once a model is loaded, simulation of the model can be started, paused, single-stepped, or reset using the play controls (Figure 4) located at the upper right of the ArtiSynth window frame. Play controls are discussed in more detail in Section 7.2.1.

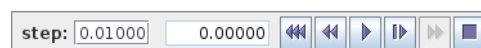


Figure 4: The ArtiSynth play controls. From left to right: step size control, current simulation time, and the reset, skip-back, play/pause, single-step, skip-forward and stop-all buttons.





Play controls are also available in the ArtiSynth timeline (Section 7). Also, hitting the ‘p’, ‘s’ and ‘r’ keys from within the viewer (Section 3.11) can be used to play/pause, single step and reset the simulation.

## 2.7 Other toolbar controls

The ArtiSynth application contains a toolbar that runs along the top of the frame. The right side contains the play controls shown in Figure 4.

When a grid is enabled in the viewer (Section 3.6), a text box appears in the center of the toolbar displaying the current grid units (Section 3.6.1).

The left side of the toolbar contains the following buttons:

	NavPanel:	Shows or hides the navigation panel (Section 4.2)
	Reset state:	Resets the simulation state at time 0 to the current state.
	Rerender:	Renders all viewers and displays.
	Enable real-time:	If pressed (the default setting), forces simulations to run no faster than real time.

If real-time is enabled (via the last button), the ArtiSynth scheduler will try to make the apparent simulation speed equal to real time. Simulations can of course run much slower than real time if they involve complex models with many degrees of freedom (such as large finite element models). However, for simpler models, real-time can also *increase* the overall simulation time, which is why the option to disable real-time is provided.

## 2.8 Saving a model

An ArtiSynth model can be saved to a file to be reloaded and used later. Selecting “Save model as ...” from the File menu will bring up a dialog that lets you select the name and directory for the model file (Figure 5). If a model file has already been specified, then one can save to it again by selecting the Save model menu item. ArtiSynth model files are text-based and are typically identified by the extension `.art`.

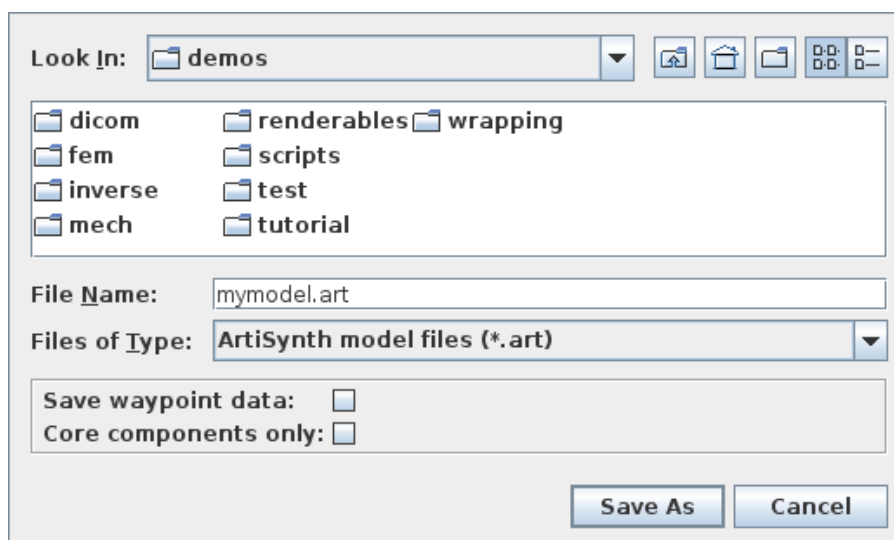


Figure 5: The save model dialog.

When using the “Save model as ...” menu item, the user may choose the following options:

### Save waypoint data:

Causes the state data for any valid waypoints (Section 7.3) to be saved (within the file) in addition to the waypoint locations. This is optional because a large number of waypoints may significantly increase the file size for models with a large state sizes.

### Core components only:

Saves only those components which are present in the main `artisynth_core` package. Any non-core components, and any other components which have a hard dependency on them, will not be written, and the user will be advised of this via a message dialog. The root model (Section 4.1) is saved as a pure instance of `RootModel`,

instead of the application-specific class that was used to build it. This means that any properties or class overrides specific to the application root model class will not be present in the saved model. The advantage to storing a model using only core components is that it can be loaded by any *other* user running the same ArtiSynth version, without needing access to any application-specific classes.

## 2.9 Setting the external classpath

As mentioned above, the classpath(s) for models declared *outside* of `artisynt_core` must be made visible to ArtiSynth so that they can be found and loaded. Detailed information on this topic is given in the section “Making external models visible to ArtiSynth” of the ArtiSynth Installation Guide.

An easy way to make classpaths visible to ArtiSynth is to add them to the *external classpath*, which is a list of top-level class folders and/or `.JAR` files containing the classes required to run external models. These may include both model classes and any external Java libraries that they require.

The external classpath is contained in a file named `EXTCLASSPATH` in the user configuration folder (Section 1.1). This file can be edited directly from ArtiSynth by selecting “External classpath ...” from the Settings menu, which will open the editing dialog shown in Figure 6.

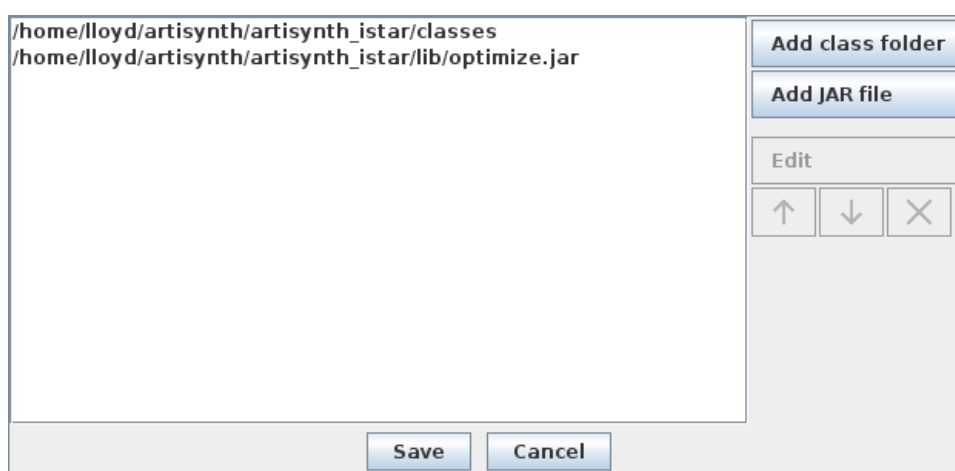


Figure 6: The external classpath editor.

The current class folders and `JAR` files are listed, one per line, in the large panel at the left. If the external classpath is empty, this panel will be blank. New class folders or `JAR` files can be added using the “Add class folder” and “Add `JAR` file” buttons at the right, which invoke file choosers appropriate to the file type. Existing entries can be selected and then edited, moved up or down in the list (using the up/down arrow buttons), or deleted (using the `X` button). When editing is complete, the updated external classpath can be saved using the `Save` button at the bottom.

ArtiSynth must be restarted for external classpath changes to come into effect.

## 2.10 The ArtiSynth working folder

ArtiSynth maintains the notion of a *working folder*, which is the default folder (or directory, in Unix parlance) under which the files used to store various types of model information are stored. This includes model files, as described above, along with other files such as those used to store waypoints, probe configurations, or probe data (Section 8).

Chooser dialogs for these files will generally be initialized to the working folder if their files have not been previously set.

The working folder is initialized to the system working folder from which the ArtiSynth application is started. Once ArtiSynth is running, it can be set by choosing “Set working folder ...” from the File menu, or by calling

```
ArtisyntPath.setWorkingFolder (file)
```

in code. When a model is saved (Section 2.8), the working folder is saved with it and restored when the model file is subsequently loaded.

### 3 The Viewer

The viewer provides interactive graphical rendering of the ArtiSynth model and permits selection of its components. A viewer is integrated into the ArtiSynth main frame; additional viewers can be created if necessary.

#### 3.1 Viewer Toolbar

Each viewer is provided with a toolbar (Figure 7) equipped with icons for controlling the viewpoint (Section 3.2) and clipping planes (Section 3.7). The toolbar for the main viewer appears vertically at the lower left of the main frame, while toolbars for additional viewers appear horizontally at the top. Each is an instance of Java's `JToolBar`, and so can be moved and docked accordingly.



Figure 7: The viewer toolbar.

#### 3.2 Viewpoint control

The viewpoint can be controlled interactively using mouse drag actions. There are three such actions:

##### Rotate

Rotates the viewpoint about the viewer center point. By default, this rotation is constrained so the viewer “up” direction remains vertical in the view plane, but this can be changed using the viewer’s `rotationMode` property (Section 3.9).

##### Translate

Translates the viewpoint in a plane perpendicular to the line of sight.

##### Zoom

Zooms in or out by moving the viewpoint along the line of sight.







The mouse button and modifier key combinations required to effect these actions depend on the application’s *mouse bindings*, which by default are set to either `ThreeButton`, `TwoButton`, or `OneButton` depending on the number of available mouse buttons. Button/key combinations for each of these is described in the following table,

Action	ThreeButton	TwoButton	OneButton
Rotate	MMB	LMB+ALT	LMB+ALT
Translate	MMB+SHIFT	LMB+ALT+SHIFT	LMB+ALT+SHIFT
Zoom	MMB+CTRL	LMB+ALT+CTRL	LMB+ALT+CTRL

where `MMB` and `LMB` denote the *middle* and *left* mouse buttons. If a mouse wheel is present, then this can also be used to execute zoom actions.

Mouse bindings can also be set explicitly by the user and alternative bindings are available; see Section 3.10.

Predetermined viewpoints can also be selected using the *align axis* button located on the viewer control bar. Clicking on this button produces a popup icon menu showing six different axis-aligned views. Each view is indicated by the two axes perpendicular to the line of sight, with the X, Y, and Z axes illustrated by red, green, and blue lines respectively. Some examples are:

	Front: Z axis up, X axis to the right.
	Back: Z axis up, X axis to the left.
	Top: Y axis up, X axis to the right.
	Bottom: Y axis down, X axis to the right.
	Left: Z axis up, y axis to the right.
	Right: Z axis up, y axis to the left.

The align axis button itself displays the most recently selected axis-aligned view, and the popup menu shows this view together with five alternates which are chosen based on the current view. Reselecting the current view will realign the viewer's viewpoint to the current view; hitting the 'v' key from within the viewer (Section 3.11) will do the same.

### 3.3 Adding additional viewers

Additional viewers can be created by selecting View > New viewer from the main menu. Each viewer provides independent viewing and selection control for the current model.

### 3.4 World coordinate axes

Hitting the 'a' key from within the viewer enables or disables drawing of the world coordinate axes. By default, these are drawn as simple lines, with the x, y and z axes colored red, green and blue, respectively, and the axis length computed automatically based on the model size.

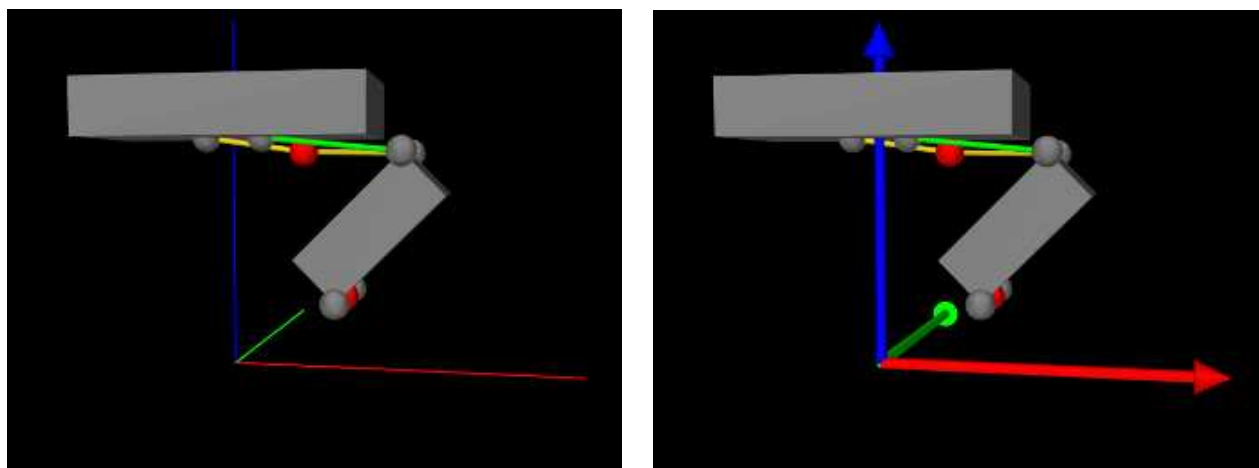


Figure 8: RigidBodyDemo with world coordinate axes drawn as simple lines (left) and solid arrows (right).

Axes can also be rendered as solid arrows by setting the `axisDrawStyle` property (Section 3.9) to `ARROW` instead of `LINE`, and the axis length can be set explicitly using the `axisLength` property (Section 3.9.1). Axes are not drawn if either `axisLength` is 0 or `axisDrawStyle` is set to `OFF`.

### 3.5 Orthographic vs. perspective projection

The user can toggle between orthographic and perspective projection by selecting View > Orthographic view or View > Perspective view from the main menu. Toggling can also be achieved using the 'o' key shortcut (Section 3.11) within the viewer.

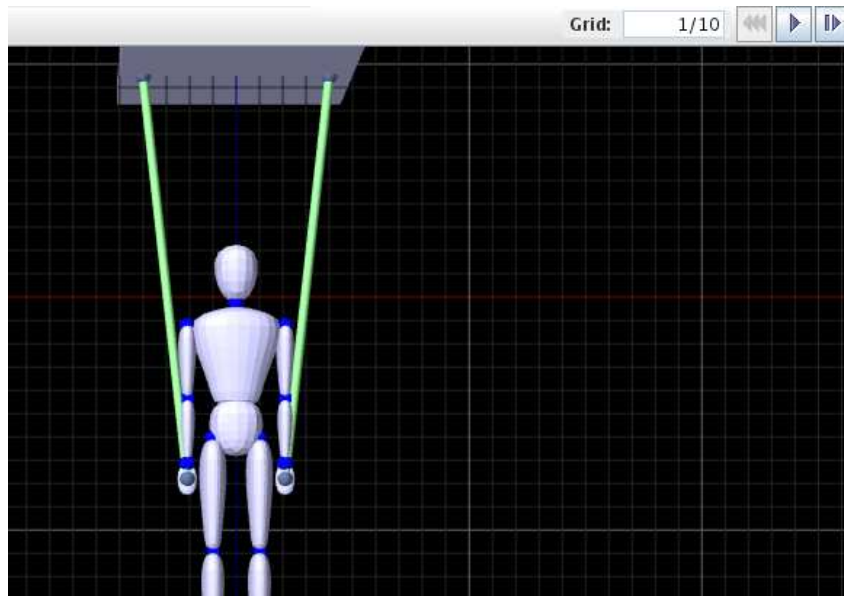


Figure 9: Viewer showing the grid.

### 3.6 Viewer grid

Hitting the ‘g’ key within the viewer enables or disables a grid (Figure 9). Grid cells are square and appear in two resolutions, with *major cells* subdivided into a number of *minor cells*. Major cells are typically rendered more brightly than minor cells. By default, the grid computes the cell sizes automatically based on the current viewer zoom-level. However, it is possible to set an explicit grid resolution (see 3.6.1).

The grid is located in the plane perpendicular to the line of sight of the most recently selected axis-aligned view. To change the grid plane, select a new axis aligned viewpoint (Section 3.2).

#### 3.6.1 Grid units

When the grid is enabled, a box labeled `Grid:` appears in the toolbar on top of the main ArtiSynth frame which gives the current resolution of the grid, displayed as  $S/N$ , where  $S$  is the size of each major grid cell and  $N$  is the number of subdivisions per cell. If there are no subdivisions, then the  $/N$  is omitted. For example, in Figure 9, this appears as `Grid: 1/10`, which means that the major grid cells have a size of 1.0 and are each divided into 10 subdivisions. The numeric value of the ratio  $S/N$  gives the minor cell size.

By default, the grid automatically resizes itself to the current viewer zoom level, choosing well-rounded numbers for the grid cell size. Auto-sizing can be enabled or disabled by right-clicking on the `Grid:` label and choosing `Turn auto-sizing` on or `Turn auto-sizing off`, as appropriate. The user can also specify an explicit value for the grid resolution by entering the desired  $S/N$  value (or just an  $S$  value) into the `Grid:` box. Specifying an explicit value will disable auto-sizing, unless  $S$  is specified as 0 or the special value `*` is entered, both of which will re-enable auto-sizing.

#### 3.6.2 Axis labeling

Hitting the ‘l’ key within the viewer enables or disables labeling of the major divisions along the horizontal and vertical axis (Figure 10). The division lines along which these labels appear are automatically adjusted so as to ensure proper label visibility, and do not necessarily correspond to the  $x$ ,  $y$ , or  $z$  axes.

It is possible to control various properties associated with axis labeling, such as which axes are labeled, and the label size and color. See the next section on Grid properties.

#### 3.6.3 Grid properties

The grid has a number of properties that can be set by right-clicking in the viewer and choosing `Set viewer grid properties` (or by right-clicking on the `Grid:` label and choosing `Set properties`). This will bring up a property dialog,

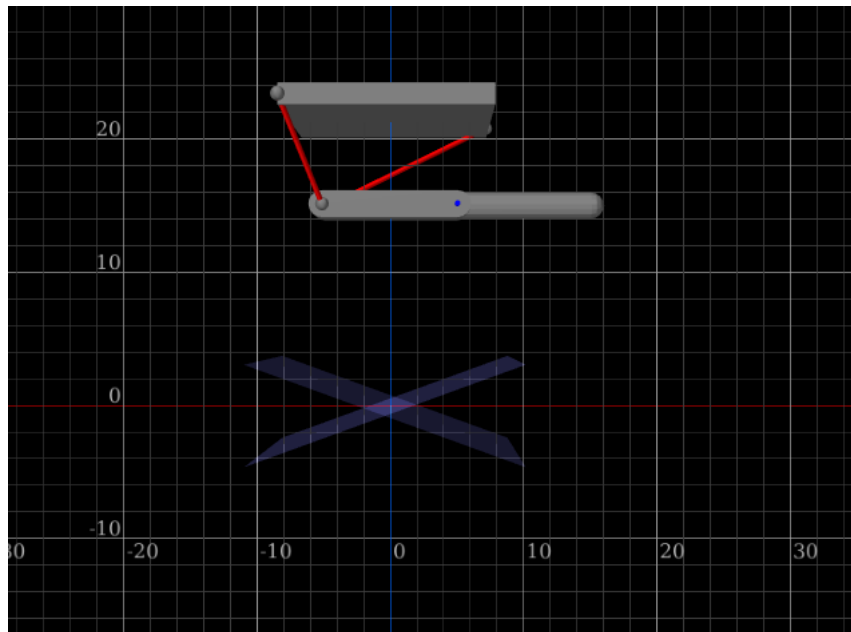


Figure 10: Viewer grid with axis labels visible.

such as that shown in Figure 11.

Properties that can be set include:

**resolution**

Grid resolution, as described above.

**autoSized**

If `true`, causes the grid resolution to be recomputed as the user adjusts the view position, orientation, and zoom.

**minCellPixels**

Minimum number of pixels that should appear in a minor cell division when autosizing.

**majorColor**

Color to use for the major axis lines.

**minorColor**

Color to use for the minor axis lines.

**xAxisColor**

Color to use for the grid line that corresponds to the world y axis, or the horizontal axis if `lockAxesToWorld` is false.

**yAxisColor**

Color to use for the grid line that corresponds to the world x axis, or the vertical axis if `lockAxesToWorld` is false.

**zAxisColor**

Color to use for the grid line that corresponds to the world z axis.

**lineWidth**

Width of the grid lines, in pixels.

**position**

Translation position of the grid, in world coordinates.

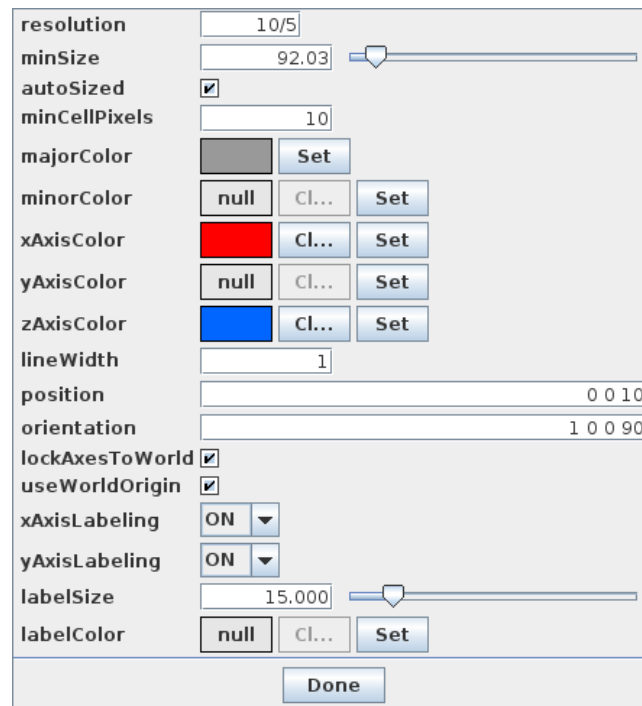


Figure 11: Dialog to control the grid properties.

**orientation**

Orientation of the grid, in world coordinates.

**lockAxesToWorld**

If `true`, forces the grid to stay aligned with the orientation and position of the world axes. In particular, the horizontal and vertical axes will always be parallel to one of the x, y, or z world axes, the grid center will be a multiple of major cell sizes from the origin, and axis labels will be set relative to the world origin.

**useWorldOrigin**

If `true`, causes the principal horizontal and vertical axes to be aligned with the world origin. Otherwise, the axes will be aligned with the grid center. This property can only be `true` if `lockAxesToWorld` is also `true`.

**xAxisLabeling**

Enables labeling of the x axis.

**yAxisLabeling**

Enables labeling of the y axis.

**labelSize**

'em' size of the label text, in pixels.

**labelColor**

If set, specifies the color used to draw the label text. Otherwise, the major axis color is used.

### 3.7 Clipping planes

The user can add clipping planes to the viewer. These are useful for restricting what is rendered and allowing a better view of interior structures, as shown in (Figure 12).

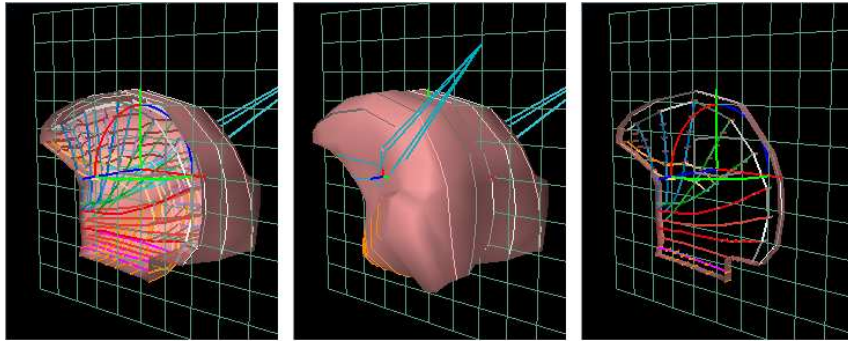



Figure 12: Clipping plane showing interior of tongue model (left), disabled (center), and in slice mode (right).

### 3.7.1 Adding and removing

To add a clipping plane, left click on the add clip plane button  located on the viewer toolbar. This will create a clipping plane located in the plane perpendicular to the current line of sight.

It will also add to the viewer toolbar a clip plane icon  for controlling the clipping plane. Right-clicking on this icon will bring up an option menu.

To delete a clipping plane, right-click on its icon and select Delete.

### 3.7.2 Moving


A clip plane is associated with a coordinate system and can be moved and/or rotated by dragging on the trans-rotate transformer located at its coordinate system origin. The clip region is the half space lying in the direction of the +z axis.

The transformer itself can be made invisible/visible by right-clicking on the clip plane icon and selecting Hide transformer or Show transformer.

### 3.7.3 Offsets

The clipping region is the half space lying in the direction of the +z axis of the plane's local coordinate system. By default, clipping is actually offset by a small distance along the +z axis, so that small objects (such as points) lying in the x-y plane remain visible. The amount of this offset is controlled by the plane's offset property, which is set to a nominal default value. To control this property directly, right-click on the clip plane icon and select Set properties. This will bring up a panel which allows the offset to be adjusted.

### 3.7.4 Enabling/disabling

Left clicking on the clip plane icon will enable/disable clipping. Disabling clipping allows the plane to be used as a regular movable grid. When clipping is disabled, the icon will change to the form .

### 3.7.5 Slicing mode

Clipping planes can be placed in a *slicing mode*, whereby half-spaces in both the positive and negative z directions are clipped. The result is a small slice about the local x-y plane (Figure 12, right). The width of this slice is controlled by the plane's offset property, as described above.

To enable or disable slicing, right-click on the clip plane icon and select Enable slicing or Disable slicing.

### 3.7.6 Other features

#### Properties

Various properties associated with the plane, such as its color, line width, cell resolution, etc., can be set explicitly by the user. To do this, right-click on the icon, select *Set properties*, and edit the resulting property panel. Most properties are the same as those described for the main viewer grid in 3.6.3.

#### Grid visibility

To make the grid invisible/visible, right-click on the icon and select *Hide grid* or *Show grid*.

#### Alignment with world axes

The clip plane can be aligned so that its normal lies along the positive or negative direction of either the x, y, or z world axes. Right-click on the icon and select the appropriate option. Clipping is performed so that the half-space lying in the direction of the normal is clipped.

#### Alignment with current line of sight

To align the clipping plane so that it is perpendicular to the current line of sight, right-click on the icon and select *Reset*.

## 3.8 Indicating 3D positions with the mouse

It is possible to use a viewer in combination with a mouse to specify the position of a 3D point in space. This is commonly employed in the editing operations described in Section 16.

To specify a point, the user left-clicks the mouse in the viewer, at the screen position located over the point's desired position. The 3D position is then determined by intersecting the ray indicated by the mouse click with some appropriate surface or plane. Typically, a plane perpendicular to the viewing direction and passing through the model's center is used. Alternatively, some interactions provide a *constrain to plane* option, which causes the ray to be intersected with a viewer clipping plane (Section 3.7), providing more precise control over the point's position. This requires that the viewer presently contain at least one clipping plane. If more than one clipping plane is present, the first one is used.

In other applications, the desired point may be known to lie on a 3D surface, in which case the position is determined by intersecting the ray with a 3D surface mesh.

## 3.9 Viewer properties

Viewers export a number of properties that control various aspects of their look and feel. Some of these can be modified collectively for all viewers by choosing "Settings > Viewers ..." from the main menu, which opens a *viewer settings* dialog (Figure 13).

Various properties can be set by this dialog, as described below. Clicking the *Save* button will save the current settings to the user's preferences (Section 11.2) so that they will be set automatically when ArtiSynth is restarted.

Properties set by the viewer settings dialog include:

#### **backgroundColor**

Color of the viewer background.

#### **selectionColor**

Color used to highlight selected items.

#### **axisDrawStyle**

Controls how world coordinate axes are drawn (Section 3.4), with *LINE* and *ARROW* specifying simple lines and solid arrows, respectively.

#### **axisRadiusRatio**

A ratio which can be used to determine the radius for an axis when the radius is not explicitly specified. The radius is computed by multiplying the ratio by the axis length. This is typically used when rendering coordinate axes as solid arrows and has a default value of 0.016.

---

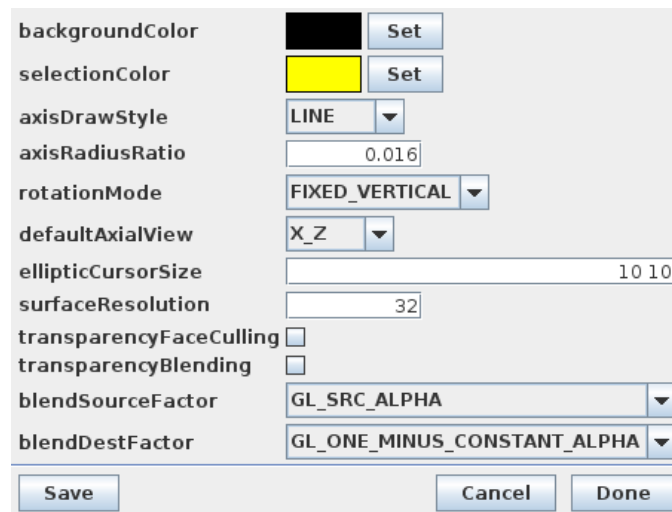


Figure 13: Viewer settings dialog.

**rotationMode**

Controls how the eye-to-world rotation is adjusted when the mouse is used to perform a rotate action (Section 3.2). The default value is `FIXED_VERTICAL`, which constrains the viewer’s “up” vector to remain vertical with respect to the view plane, at the expense of preventing the eye-to-world rotation from being adjustable to an arbitrary value. Alternate values include `CONTINUOUS`, which enables a track-ball type rotation that does allow arbitrary adjustment of the eye-to-world rotation, and `OFF`, which disables rotation adjustment.

**defaultAxialView**

Sets the default axis alignment indicating how the 3D world axes correspond to the horizontal “right” and vertical “up” view plane directions. Each setting takes the form  $R\_U$ , where  $R$  is the world axis pointing right and  $U$  is the axis pointing up, and  $R$  and  $U$  can each be either  $X$ ,  $Y$ ,  $Z$ ,  $NX$ ,  $NY$ , or  $NZ$ , indicating the positive or negative  $x$ ,  $y$  or  $z$  axis.

**ellipticCursorSize**

Size of the elliptic cursor (Section 4.3.2) in the horizontal and vertical directions. The default value is (10,10).

**surfaceResolution**

Controls the number of faces or segments used for rendering built-in curved primitives, such as cylinders and spheres. For cylinders, it controls the number of sides, while for spheres it controls the number of longitudinal slices. A larger number produces a smoother effect at increased graphical cost. The default value is 32.

For OpenGL based viewers, the following properties are also provided to control how transparency is rendered:

**transparencyFaceCulling**

Enables or disables face culling when rendering transparency.

**transparencyBlending**

Enables or disables transparency blending (via `glEnable()` or `glDisable()` using `GL_BLEND`) when rendering transparency.

**blendSourceFactor**

Specifies the first (source) argument to `glBlendFunc()`.

**blendDestFactor**

Specifies the second (destination) argument to `glBlendFunc()`.

### 3.9.1 Viewer-specific properties

Viewer properties can also be set on a per-viewer basis. To do this, invoke the context menu (usually right-click) in the viewer when nothing is selected, and choose **Set viewer properties**. Individual properties include all those described above (except with `defaultAxialView` renamed to `axialView`), together with the following additional properties:

#### **axisLength**

Axis lengths used to render the world axes (Section 3.4).

#### **viewControlMask**

If set to `ALONG_X_ONLY` or `ALONG_Y_ONLY`, mouse-based view control inputs are restricted to those in the x or y screen directions, respectively. This makes it easier to limit changes in the view point. The default value is `NONE` (no restriction).

#### **eye**

Location of the eye position, in world coordinates.

#### **center**

Location of the viewing frustum center, in world coordinates.

Setting a viewer-specific property, such as the background color, will generally cause it to have a value that differs from its counterpart in the viewer settings (because the value was set for only a single viewer). On the other hand, properties set in the viewer settings will be applied to *all* open viewers.

## 3.10 Mouse Bindings

The ArtiSynth GUI was originally designed for a three-button mouse, in which the left button is used for selection, the middle button controls the viewpoint, and the right button is used to activate the context menu. These are used in conjunction with the modifier keys `SHIFT` and `CTRL` to effect different actions.

For systems that do not have a three-button mouse, ArtiSynth by default detects the number of mouse buttons and adjusts the mouse bindings so that the `ALT` key emulates the middle button and the `META` key emulates the right button.

The `META` key is usually associated with either the `COMMAND` key (Mac) or the `WINDOWS` key.

Mouse bindings can also be explicitly set by the user, by choosing “Settings > Mouse ...” from the main menu, which opens a *mouse bindings* dialog (Figure 14). This allows the user to change the bindings, and also for any given binding describes the button/key combinations to effect various actions. If the “Auto detect” checkbox is selected, then the bindings are determined automatically from the number of available mouse buttons. Unchecking this box allows the bindings to be set explicitly using the Bindings selector. The dialog also allows control of the scale factor used for mouse wheel zoom actions.

Clicking the **Save** button in the mouse bindings dialog will save the current bindings to the user’s preferences (Section 11.2) so that they will be set automatically when ArtiSynth is restarted. Mouse bindings can also be specified explicitly at startup using the `-mousePrefs <bindings>` command line option.

Currently, there are five bindings available:

#### **ThreeButton**

Default bindings for a three-button mouse.

#### **TwoButton**

Default bindings for a two-button mouse. The middle mouse button is emulated with the `ALT` key.

#### **OneButton**

Default bindings for a one-button mouse. The middle and right mouse buttons is emulated with the `ALT` and `META` keys.

**Auto detect** ☒

**Bindings** ThreeButton ▼

*Viewpoint control:*

Rotate view

Translate view

Zoom view

*Component selection:*

Select components

Multiple selection

Elliptic deselect

Resize elliptic cursor

Context menu

*Manipulator controls:*

Move dragger

Dragger constrain

Dragger reposition

*(LMB, MMB, RMB = left, middle, right mouse buttons)*

**Wheel zoom scale**

**Save** **Cancel** **Done**

Figure 14: Mouse bindings dialog.

**Laptop**

Legacy bindings for a two-button mouse.

**Mac**

Legacy bindings for a Mac type one-button mouse.

Tables showing the button and modifier key combinations that effect different actions for each of these are given below, with LMB, MMB, and RMB denoting the left, right and middle mouse buttons. Actions marked with an asterisk (\*) are drag actions which can have their modifier keys invoked or removed during a drag operation. Button/key combinations for ThreeButton, TwoButton, and OneButton are:

Action	ThreeButton	TwoButton	OneButton
<b>Viewpoint control (Section 3.2)</b>			
Rotate view	MMB	LMB+ALT	LMB+ALT
Translate view	MMB+SHIFT	LMB+ALT+SHIFT	LMB+ALT+SHIFT
Zoom view	MMB+CTRL	LMB+ALT+CTRL	LMB+ALT+CTRL
<b>Component selection (Section 4.3)</b>			
Select components	LMB	LMB	LMB
Multiple selection	LMB+CTRL	LMB+CTRL	LMB+CTRL
Elliptic selection	LMB	LMB	LMB
Elliptic deselection*	LMB+SHIFT	LMB+SHIFT	LMB+SHIFT
Resize elliptic cursor	LMB+SHIFT+CTRL	LMB+SHIFT+CTRL	LMB+SHIFT+CTRL
Context menu	RMB	RMB	LMB+META
<b>Manipulator controls (Section 5.2)</b>			
Move dragger	LMB	LMB	LMB
Dragger constrain*	LMB+SHIFT	LMB+SHIFT	LMB+SHIFT
Dragger reposition*	LMB+CTRL	LMB+CTRL	LMB+CTRL

while those for Laptop and Mac are:

Action	Laptop	Mac
Viewpoint control (Section 3.2)		
Rotate view	LMB	LMB+ALT
Translate view	LMB+SHIFT	LMB+ALT+SHIFT
Zoom view	LMB+ALT	LMB+ALT+META
Component selection (Section 4.3)		
Select components	LMB+CTRL	LMB
Multiple selection	LMB+SHIFT+CTRL	LMB+META
Elliptic selection	LMB+CTRL	LMB
Elliptic deselection*	LMB+SHIFT+CTRL	LMB+SHIFT
Resize elliptic cursor	LMB+SHIFT+CTRL	LMB+SHIFT+CTRL
Context menu	RMB	LMB+CTRL
Transformer control (Section 5.2)		
Move dragger	LMB	LMB
Dragger constrain*	LMB+SHIFT	LMB+SHIFT
Dragger reposition*	LMB+ALT	LMB+ALT

### 3.11 Keyboard shortcuts

When the viewer has the keyboard focus, the following key shortcuts are available:

Key	Operation
q	quit ArtiSynth
t	toggle time line visibility
z	undo last command
Play controls (Section 2.6):	
p or SPC	play/pause
s	single step
r	reset
Viewer controls:	
v	reset view (Section 3.2)
o	toggle orthographic/perspective view (Section 3.5)
a	toggle visibility of axes showing world coordinates
g	toggle viewer grid (Section 3.6)
l	toggle viewer grid labels
Selection and transformer (Sections 4.3 and 5.2):	
ESC	select parent of last selection
c	clear selection
d	reset elliptic cursor size to default
w	set current transformer frame to world coordinates
b	set current transformer frame to body/local coordinates

## 4 Component Navigation and Selection

An ArtiSynth model is composed of a hierarchical arrangement of model components (each of which implements the interface `ModelComponent`), some of which may themselves be models. The graphical interface allows users to navigate this hierarchy and select individual components. Selected components can then be edited, or have specific properties modified or attached to probes or control panels.

### 4.1 The component hierarchy

An example component hierarchy is shown in Figure 15. At the top is a *root model* (class `RootModel`), in this case named `Rigid Body Spring`. The root model in turn contains a list of models, one of which is a mechanical model named `msmod`, which here contains particles and rigid bodies.

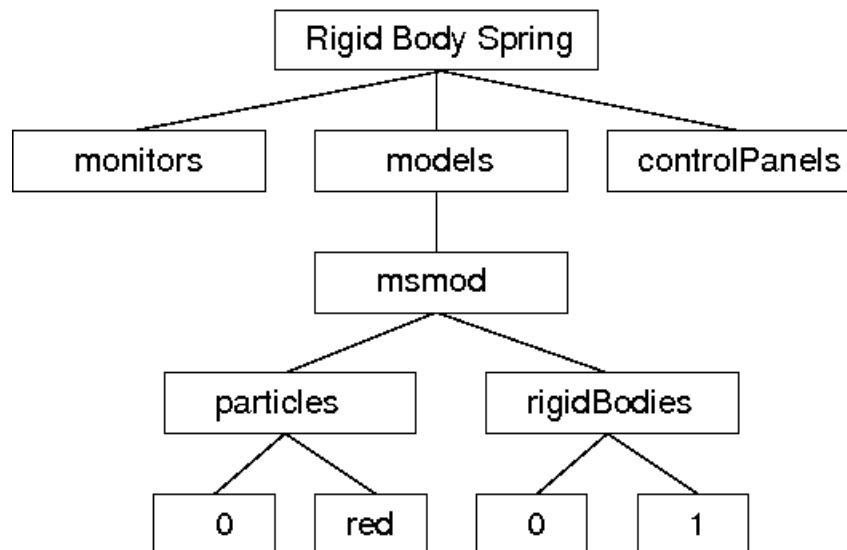


Figure 15: A sample component hierarchy.

It is important to note that in the component hierarchy, any collection of components is itself a component (usually an instance of `ComponentList`). This provides automatic “grouping” of components of like type, but does introduce additional levels into the hierarchy. Hence the particle `red` is a child not of `msmod`, but rather the component list `particles`.

#### 4.1.1 Component names and numbers

Model components may be assigned a string name; at the time of this writing names may not begin with a digit, have zero length, contain the characters ‘.’ or ‘/’, or equal the reserved word `this`. Components which do not have an assigned name are called *nameless*.

All components have a *number*, even if they do not have a name. The number is assigned automatically when the component is added to the parent, and is guaranteed to be persistent until the component is removed from the parent.

#### 4.1.2 Component path names

The names and/or numbers of a component’s ancestors can be used to form a *component path name*. This path has a construction completely analogous to Unix file path names, with the ‘/’ character acting as a separator. Absolute paths start with ‘/’ and begin with the root model. Relative paths omit the leading ‘/’ and can begin lower down in the hierarchy. The absolute path name of the `red` particle in Figure 15 would be

```
/Rigid Body Spring/models/msmod/particles/red
```

For nameless components in the path, their numbers can be used instead:

```
/Rigid Body Spring/models/msmod/rigidBodies/1
```

Numbers can be used even for components that have names. Hence a path name consisting only of numbers, as in

```
/0/0/0/3/1
```

is legal, although it most likely to appear only in machine-generated output.

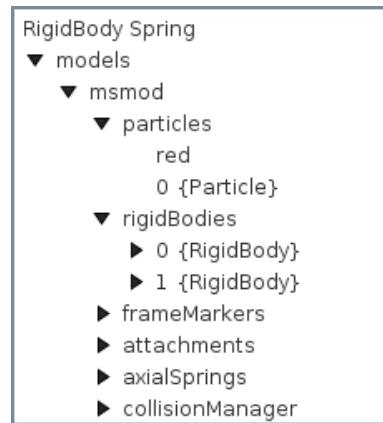


Figure 16: An typical navigation panel display.

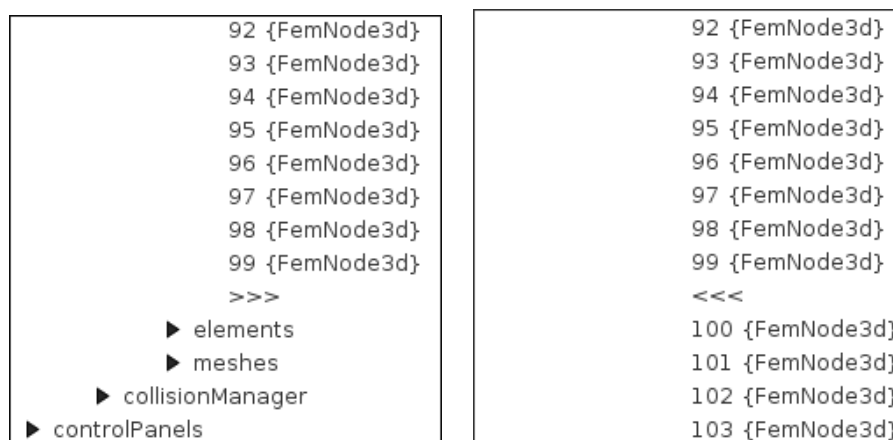


Figure 17: Expansion of nameless components in the navigation panel.

## 4.2 Navigation panel selection


A navigation panel in the main ArtiSynth frame allows direct navigation of the component hierarchy. The panel can be open or closed by clicking on the main toolbar icon .

Figure 16 shows an navigation panel containing a superset of the hierarchy diagrammed in Figure 15.

Left clicking on any component in the navigation panel selects that component. Clicking while pressing the **CTRL** key (or the **CMD** key on some platforms, such as Mac) allows selection of multiple components. Clicking while pressing the **SHIFT** key allows selection of a range of components.

### 4.2.1 Large numbers of nameless components

In some cases, such as finite element models, the number of child components can be very large (on the order of thousands). In order to keep the navigation panel size manageable, the number of nameless children displayed is limited to a set number (currently 100). If the number of nameless children exceeds this number, the display will be augmented with an expand icon **>>>**. Clicking on this will expand the display to include all nameless components, and the expand icon will be replaced by a contract icon **<<<**. Clicking on the contract icon will cause the extra nameless components to be hidden again. This is illustrated in Figure 17.

## 4.3 Viewer selection

Components that are rendered in the viewer can generally be selected by variety of methods (the exception is for a few renderable components that do not support selection). These methods include *click*, *box*, and *elliptic* selection. The top

two icons in the selection toolbar at the left of the ArtiSynth frame control the current selection method. In addition, hitting the 'c' key from within the viewer (Section 3.11) clears the current selection.

#### 4.3.1 Click and box selection

Click and box selection are enabled by the arrow icon at the top of the selection toolbar:



Click selection involves left clicking on a component, causing it to be selected. Selection of multiple components is enabled by left clicking with a modifier key, which is usually `CTRL` but may be different for some legacy mouse bindings (Section 3.10).

Click selection selects only those components which are actually visible to the viewer; components which are hidden cannot be selected this way.

Box selection is effected by left-clicking and dragging in the viewer, causing the selection of all components rendered within the resulting drag box. Because this often results in the selection of more components than desired, it may be useful to employ a selection filter (Section 4.3.3). Any components within the drag box which are already selected will be deselected.

Box selection acts on *all* (filtered) components within the view frustum defined by the drag box, including those which are hidden from view.

#### 4.3.2 Elliptic selection

Elliptic selection is enabled by the elliptic icon near the top of the selection toolbar:



This causes an additional elliptic cursor (which defaults to a circle) to be drawn around the mouse cursor. Selection is effected by dragging, and causes all visible objects within the ellipse to be selected. The selection process is cumulative, with subsequent drags selecting additional components. As with all selection operations, a filter can be set to restrict the components that are selected (Section 4.3.3). Generally, the drag select operation requires no modifier keys, although it may with some legacy mouse bindings (Section 3.10).

It is also possible to *deselect* components in the same way, by using the `SHIFT` modifier key to cause drag operations to cumulatively deselect components.

Elliptic selection selects only those components which are actually visible to the viewer; components which are hidden cannot be selected this way.

The elliptic cursor used for selection can be resized, either interactively, or by setting the `ellipticCursorSize` property of the viewer (Section 3.9). To interactively change the cursor size, initiate a drag operation with the `CTRL` and `SHIFT` modifiers. Finally, the 'd' key shortcut within the viewer will cause the cursor to be reset to its default size.

#### 4.3.3 Selection filtering

It is possible to limit viewer selection to components of a specific type. This can be done using the selection filter widget at the bottom left of the main ArtiSynth frame Figure 18.

To enable filtering, type into the widget text box the class name of the component type you wish to restrict filtering to. It is generally only necessary to enter the leaf name of the class (e.g., `Particle` or `AxialSpring`), and the system will then find the full class name by searching the ArtiSynth class path.

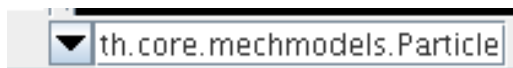


Figure 18: The selection filter widget.



Figure 19: The selection display widget.

Once filtering is enabled, only components which are instances (including subclasses) of the specified type will be selectable.

Previously selected filters can be recalled using a history list accessible using the leftmost arrow button on the selection widget.

To remove selection filtering, enter the special filter \*, either by typing this in the text box, or using the history list.

#### 4.4 Selection display

The selection display Figure 19 at the bottom of the main ArtiSynth frame shows the full path name of the last component added to the selection list. This is useful for identifying components in detail.

If no components are selected, then the selection display is blank.

The selection display is useful for disambiguating situations where it is not clear what component we have actually selected in the viewer. For example, FEM models keep their surface mesh contained within a descendant component. Selecting the surface mesh will cause this container component to be selected and highlighted, making it *appear* as though the FEM model itself is selected rather than the container. Checking the selection display makes it clear what component has actually been selected. If desired, one can easily navigate to one of the ancestor components using parent selection, as described in the next section.

#### 4.5 Selecting parent and ancestor components

Sometimes, when you select a component, you actually want to select one of its ancestor components.

There are several ways to do this:

1. Hit the escape (ESC) key within the viewer window. This will select the parent of the currently selected component. Hitting escape repeatedly is a fast way to proceed up the component hierarchy.
2. Click on the “up” arrow located at the left of the selection display (Figure 19). This will also select the parent of the currently selected component.

Parent selection is particularly useful in the commonly occurring situation where a composite component is not rendered and therefore not selectable in the viewer. For instance, suppose we wish to select a FEM model. One can select any renderable descendant of the model, such as a node, element, or its surface mesh (if displayed), and then use repeated parent selection until the model itself is selected.

#### 4.6 Highlighting selected components

Selected components are rendered in the viewer using a special selection color (yellow at the time of this writing). It is important to note that descendants of a selected component are **not** presently rendered in any special way. For instance, if an FEM is selected, its nodes and elements will be rendered normally.

While this has the potential to be confusing, we have not yet found this to be problematic, as the navigation panel and selection display provide alternative indicators as to what is currently selected.

## 4.7 Selecting FEM nodes

A specialized tool is available for selecting FEM nodes, because it is often necessary to identify collections of nodes for connecting to rigid bodies or other FEM models, or to set localized properties and/or boundary conditions.

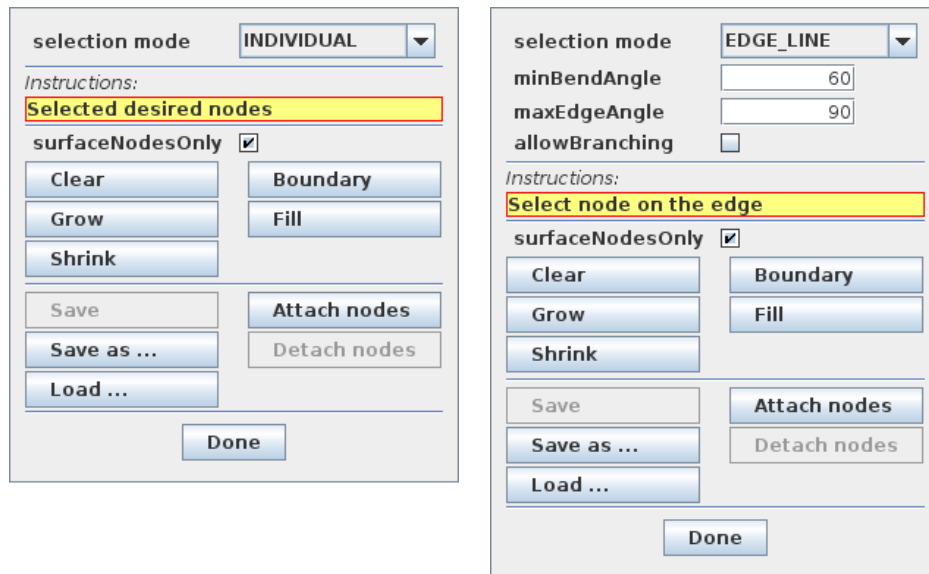


Figure 20: FEM node selection tool with different selection mode values. Left: selection mode set to the default value `INDIVIDUAL`. Right: selection mode set to `EDGE_LINE`, with fields for adjusting the `EDGE_LINE` behavior appearing immediately below the mode selector.

To open the node selection tool, first select the FEM model containing the nodes, and then invoke the context menu (usually via right-click) and choose “Select nodes ...”. A node selection tool will appear, as shown in Figure 20. From top to bottom, this contains several controls:

1. A selection mode chooser that specifies how nodes are selected, possibly followed by mode-specific fields to adjust the selection behavior;
2. An instruction box describing what action the user should take;
3. A `surfaceNodesOnly` check box that allows selected nodes to be restricted to the FEM model’s surface;
4. Buttons for clearing, growing or shrinking the selection, reducing the selection to its boundary, or for filling it in;
5. Buttons for saving/loading a list of the selected nodes to/from a file, or for attaching or detaching selected nodes from other bodies.

While a node selection tool is open, GUI selections are restricted to certain components, usually nodes of the indicated FEM model. If `surfaceNodesOnly` is checked, selection is restricted to the FEM surface nodes. Selection is also held in a “multiple component” mode, equivalent to holding down the multiple selection key (usually `CTRL`), so that the selection can be easily edited and is not reset with each selection action.

The different selection modes and their parameters include:

### INDIVIDUAL

Nodes are selected directly by the user, using click, box or elliptic selection (Sections 4.3.1 and 4.3.2). Selection is held in a “multiple component” mode, regardless of whether the “multiple selection” modifier key is pressed.

### PATCH

Nodes are selected, on the mesh surface, within a *patch* region defined by a set of faces for which the *bend angle* between adjacent faces is less than or equal to a specified *maximum bend angle*. The bend angle is defined as the

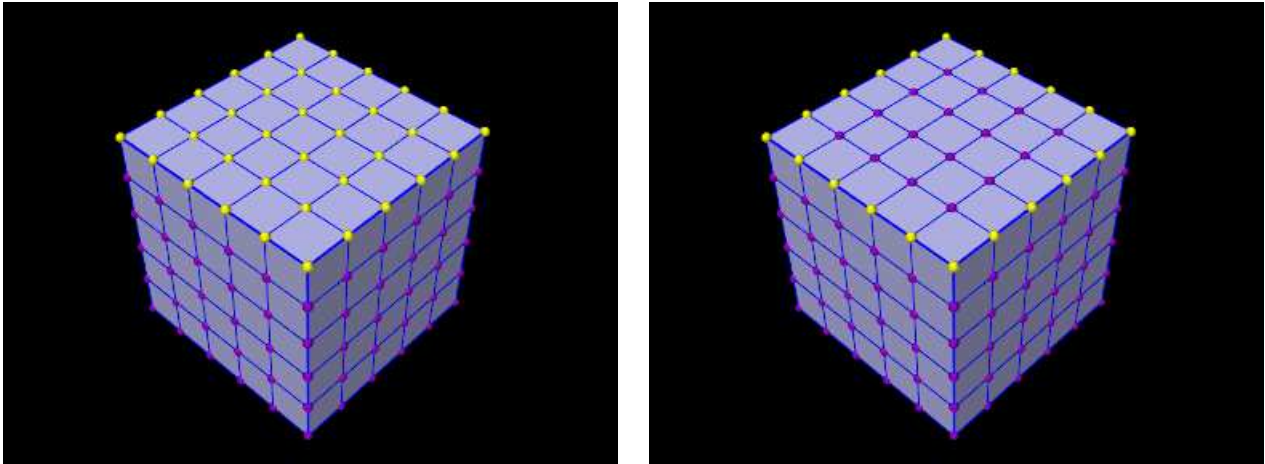


Figure 21: Left: selection of nodes on the top face of a FEM cube using `PATCH` selection. Right: reduction of the selected region to its border after clicking on the `Boundary` button.

absolute value of the angle between two faces along their common edge, such that parallel faces have a bend angle of  $0^\circ$  and perpendicular ones have a bend angle of  $90^\circ$ . Within the tool, the maximum bend angle is specified (in degrees) by the field `maxBendAngle`. To find the patch nodes, the user selects a node within the desired patch area. Starting with the node's adjacent faces, the patch is then grown outwards to find all neighboring faces such that the bend angle between them does not exceed `maxBendAngle` (Figure 21, left).

### EDGE\_LINE

Nodes are selected, on the mesh surface, along an *edge line* defined by faces meeting at a bend angle greater than or equal to a specified *minimum bend angle*. Within the tool, this minimum angle is specified (in degrees) by the field `minBendAngle`. To find edge line nodes, the user selects a node lying along the desired edge line. The line is then followed from node to node, by locating edges with a large enough bend angle.

`EDGE_LINE` selection is illustrated in Figure 22, and the following fields adjust its behavior:

`minBendAngle`

Minimum bend angle, described above.

`maxEdgeAngle`

Specifies (in degrees) the *maximum edge angle* between two adjacent edges along the edge line. Line following is terminated when the angle between the current edge and the next edge exceeds this threshold.

`allowBranching`

If selected, allows the edge line to branch in different directions. Otherwise, edges are followed sequentially so as to minimize the edge angle between them.

### DISTANCE

Nodes are selected based on being within a prescribed distance of a polygonal mesh, which may be the surface mesh for a rigid body or FEM model, or some other mesh component. The distance threshold is specified by the field `distance`. To select nodes, the user may select any mesh component containing a polygonal mesh. The user may also select a rigid body or FEM model, which will cause the body's surface mesh to be used for the distance calculation.

The following fields adjust the behavior of `DISTANCE` selection:

`distance`

Distance threshold, described above.

`useSignedDistance`

If selected, causes the distance computation for closed meshes to be *signed*, so that nodes inside the mesh will have a negative distance. This means that for a non-negative distance threshold, nodes inside the mesh will always be selected.

`surfaceNodesOnly`

If checked, the selection is restricted to the FEM model's surface.

## MINIMUM\_PATH

In this mode, the user selects an individual node, and the tool then selects a *minimum path* of nodes between it and the previously selected node. This is particularly useful for “drawing” the boundary of a nodal region on the FEM model’s surface.

If `surfaceNodesOnly` is checked, the path will be restricted to the FEM surface; otherwise, the path may proceed through the FEM interior.

When forming the path, the tool chooses from adjacent nodes based on the topology induced by the FEM elements. This can sometimes lead to a path that is less minimal than if nodes could be chosen arbitrarily, particularly for quadratic hex, wedge and pyramid elements.

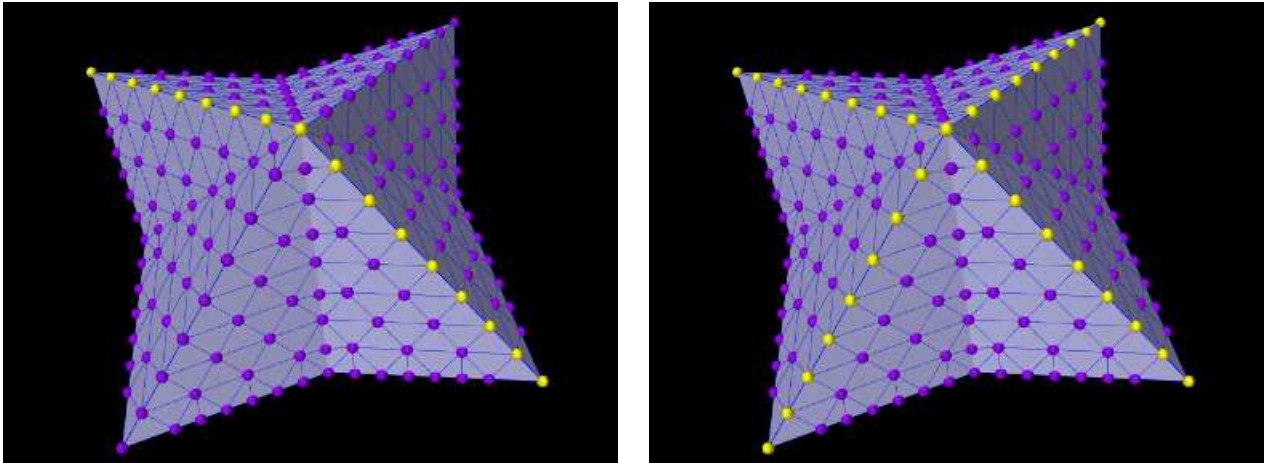


Figure 22: Node selection of ridges on an FEM using `EDGE_LINE` selection, with `maxBendAngle` and `maxEdgeAngle` set to  $30^\circ$  and  $120^\circ$ , respectively. The left and right images show the resulting selection with branching disabled and enabled.

The user is free to change modes during the selection process. Because selection is held in multiple selection mode, it is easy to edit the selection. In particular, `INDIVIDUAL` selection can be used for fine editing of selections obtained using `PATCH`, `EDGE_LINE`, `DISTANCE`, or `MINIMAL_PATH`.

Below the `surfaceNodesOnly` check box, the tool contains a number of buttons:

### Clear

Unselects all the nodes.

### Grow

Enlarges the current selection to include adjacent nodes.

### Shrink

Reduces the current selection by removing nodes on the boundary.

### Boundary

Reduces the current selection to include *only* its boundary nodes (Figure 21, right).

### Fill

Fills a region in the current selection. After clicking this button, the user selects a node inside the desired region. The selection is then grown outward from the selected node until it reaches the region’s boundary.

### Attach nodes

Attaches the selected nodes to another body which is either a rigid body or an FEM model. The body may be chosen by selecting it directly, or by selecting one of its component meshes. Nodes which are currently attached are *not* attached to the body.

**Detach nodes**

Detaches all selected nodes from any body or particle that they are currently attached to.

**Save**

Saves the nodes to a file that has been previously specified using either “Save as ...” or “Load ...”.

**Save as ...**

Saves the nodes to a user-specified text file. The format consists simply of integer node numbers (with respect to the selection tool’s FEM model), separated by white space. More details on the file format are given in the section “Selecting nodes in the viewer” of the [ArtiSynth Modeling Guide](#).

**Load ...**

Loads the selected nodes from a text file, the format for which is described under “Save as ...”.

## 5 Model Manipulation

Various tools located within the selection toolbar at the left of the main ArtiSynth frame allow models to be manipulated in various ways. These include modifying component locations, orientations, and geometry using the transformer tools (Section 5.2), exerting point forces on selected components using the pull controller (Section 5.3), and adding marker points to certain components types (Section 5.4).

The behavior of these tools is somewhat context dependent. For example, the transformer tools will only transform those *transformable* components which implement the [TransformableGeometry](#) interface. The behavior may also vary depending on whether or not simulation is in progress.

### 5.1 Dragger fixtures

The transformation tools employ the dragger fixtures shown in Figure 23, which allow 3D geometrical transformations to be performed within the viewer.

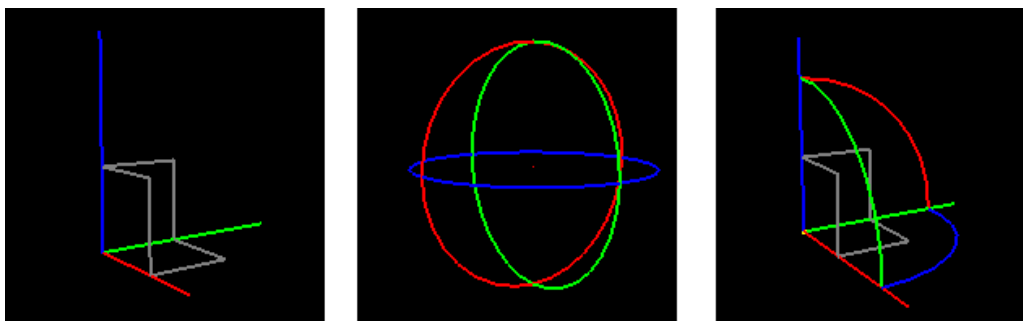


Figure 23: Dragger fixtures: translator, rotator, and transrotator.

**Translator**

Effects a translation. The x, y, and z axes are indicated by red, green, and blue lines. Dragging any line causes a one-dimensional motion along the associated axis. Dragging one of the boxes causes a two-dimensional motion in the associated plane.

**Rotator**

Effects a rotation. Rotation about the x, y, and z axes is indicated by red, green, and blue circles. Selecting and dragging along one of these circles produces a rotation about the corresponding axis.

**TransRotator**

Combines the translator and rotator into a single tool. One difference is that the axes of the transRotator move with any rotation, and so operations are done with respect to the transRotator coordinate frame at the beginning of the drag.

Under the default mouse bindings, the basic drag operations involving these fixtures are invoked using the left mouse button with no modifier keys. Additional modifier keys allow constrained transformation or repositioning of the fixture, as described below.

## 5.2 Transformer tools

A number of transformer tools use the dragger fixtures described above to translate, rotate, and scale components. Once a tool is activated, then selecting one or more transformable components will cause the corresponding dragger fixture to appear in the viewer at the components' location. If a single component is selected and that component is associated with a coordinate frame (by implementing the [HasCoordinateFrame](#) interface), then the dragger's initial position and orientation are aligned with this coordinate frame. Otherwise, the dragger is initially placed at the center of the selected components with an orientation aligned to world coordinates.

The initial dragger orientation can be adjusted in the following ways:

1. To request that a dragger's initial orientation is *always* aligned with world coordinates, choose "Interaction ..." from the Settings menu and set `initDraggersInWorld` to `true`.
2. Conversely, when transforming a set of point-like objects which lie either within a plane or along a straight line, it can be useful to align the dragger's orientation to the points. To request this behavior, choose "Interaction ..." from the Settings menu and set `alignDraggersToPoints` to `true`.

The transformer tools include:



Translation:

Translates selected components using the translator dragger.



Rotation:

Rotates selected components using the rotator dragger.



TransRotation:

Translates and rotates selected components using the transRotate dragger. The transformation reference frame moves with the tool.



Constrained translation:

Translates selected components using the translate dragger while ensuring that they are constrained to remain on a surface mesh. Only components with an associated surface mesh (such as `FrameMarkers` attached to a `RigidBody`) can be transformed this way.



Scaling:

Scales selected components using the transrotator fixture. Instead of translating, translational drag operations scale the component along the x, y, or z axes, or in the x-y, y-z, or z-x planes. Rotational operations, if used in conjunction with an appropriate modifier key, can be used to change the orientation of the scaling axes, as described in [5.2.2](#).

### 5.2.1 Constrained transformation

Under the default mouse bindings, pressing the `SHIFT` modifier key causes drag operations to be constrained to discrete step sizes. Rotation operations are constrained to intervals of five degrees, while translation operations are constrained to either the grid spacing (if a grid is selected, see [3.6](#)), or to a suitable well-rounded number depending on the viewer's zoom level.

### 5.2.2 Transformer repositioning

Under the default mouse bindings, pressing the `CTRL` modifier key causes the dragger fixture to move independently of the selected objects. This allows the transformer frame to be changed relative to the selected objects being manipulated; this is particularly useful for changing the orientation of the scaling directions in the scaling tool.

If a single object is being transformed (as opposed to a group of selected objects), the application will remember the modified transformer frame if the object is deselected and then reselected. The default frame can be restored by hitting the 'u' key (Section 5.2.3).

### 5.2.3 Changing the transformer base frame

By default, a transformer is assigned a local coordinate frame for the object(s) that it is positioning, based on either the object's own body frame (if it has one), or the objects' bounding box. This frame will then move with the transformer, and may also move relative to the object(s) if the transformer is repositioned (Section 5.2.2).

Sometimes, it may be desirable to explicitly reset the transformer's frame. This may be done using the following shortcut keys in the viewer (while a transformer tool is being used and objects are selected):

w

Set the transformer frame to the world coordinate system, allowing subsequent transformations to be performed in world coordinates;

b

Reset the transformer frame to the local frame for the object(s), based on either the object's body frame or the objects' bounding box. For single a object, any changes to the transformer frame that were made using the `CTRL` modifier (Section 5.2.2) will be restored.

u

Reset the transformer frame to the *default* local frame for the object(s), based on either the object's body frame or the objects' bounding box. For single a object, any changes to the transformer frame that were made using the `CTRL` modifier (Section 5.2.2) will be removed.

### 5.2.4 Flipping transformer axes forward

Sometimes, a user may discover that a transformer's frame is inconveniently oriented, with one or more axes directed away from the viewer window, obscuring them and making the transformer difficult to use. This can be mitigated by hitting the 'f' key in the viewer, which will cause the axes of any active transformer to be flipped so that they are directed toward the viewer window (Figure 24).

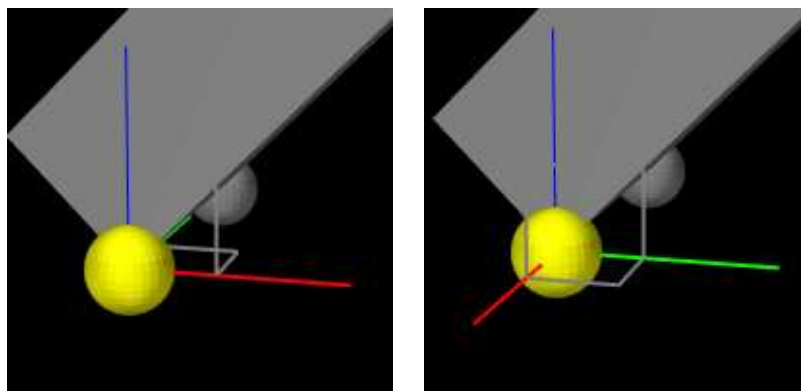


Figure 24: Translational transformer applied to a frame marker, with axes initially obscured (left) and then brought forward after hitting the 'f' key (right).

### 5.2.5 Resizing transformers

ArtiSynth will try to automatically set a transformer's dragger fixture to an appropriate size. In some cases, the user may wish to change this size, which may be done using the following shortcut keys in the viewer (while a transformer tool is being used and objects are selected):

CTRL u **or** UP\_ARROW

Increases the transformer size.

CTRL d **or** DOWN\_ARROW

Decreases the transformer size.

The default size can be restored by hitting the 'u' key (Section 5.2.3).

## 5.3 Pull manipulation



Pull tool:

Pulls on certain components using a spring-like force when simulation is running.

Selecting the *pull* tool allows a user to interactively apply a spring-like force to certain component types by clicking on it and then dragging (Figure 25). Double clicking on the component adds a *pull point* that persists between mouse clicks; to remove the pull point, double click on the viewer background.

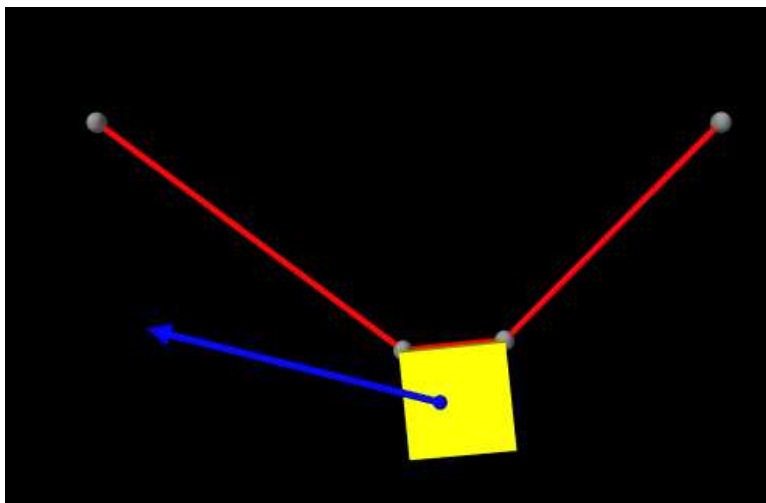


Figure 25: Applying pull manipulation (blue arrow) to a rigid body attached to a multi-point spring.

The pull tool works on points, rigid bodies, FEM models, or any other component that implements the [PointAttachable](#) interface and has a surface mesh. Pull manipulation is only effective when simulation is running. It works by adding a special `PullController` to the current root model's set of controllers. When attached to the root model, the controller attempts to estimate an appropriate spring stiffness based on the overall mass and dimensions of the first underlying `MechModel`.

If necessary, the pull tool's stiffness setting can also be adjusted manually by selecting `PullController > properties` in the Settings menu. Render properties for the pull controller can be set from this menu also.

## 5.4 Marker tool



Marker tool:

Adds point markers to certain components.

Selecting the *marker* tool allows a user to interactively add point markers to various components in the model.

Markers can be added by double-clicking on rigid bodies, FEM models, and other components that implement the [IsMarkable](#) interface.

Markers are typically added to a component's surface mesh, and in such cases it is necessary that this surface mesh be visible. For FEM models in particular, the `surfaceRendering` property should be set to something other than `None`. To ensure this is the case, select the FEM model, right-click and choose "Edit properties ...", and examine the setting for `surfaceRendering`. An FEM model can sometimes *appear* to have a visible surface mesh, even if it doesn't, if its `elementWidgetSize` property is close to 1.

The markers themselves are added to the current root model at a location that depends on the component being marked:

- **Rigid bodies:** marker is added to the `frameMarker` list for the body's parent `MechModel`;
- **FEM models:** marker is added to the FEM model's `marker` list;
- **IsMarkable components:** marker is added to the root model's `marker` list (which is created on demand if needed).

In order for added markers to be visible, the component list into which they are placed needs to be visible, with its point rendering properties set to appropriate values. That usually means that `pointStyle` is set to `SPHERE` or `CUBE`, with `pointRadius` set to a value commensurate with the model's dimensions, *or* `pointStyle` is set to `POINT`, with `pointSize` set to a sufficiently large value in pixels. If not set within the list itself, the point rendering properties will be inherited from ancestor components. For example, if a `MechModel`'s point render properties are set for good visibility, then all points within subcomponents will be visible unless these settings have been overridden at a lower point in the hierarchy.

If markers are not appearing when using the marker tool, use the navigation panel to open the component list to which the markers are added (as described above). Verify that markers are actually being appended to the end of the list. If they are not, ensure that the component's surface mesh is visible. If they are, select the list itself, right-click, and choose "Edit render props ...". Check that `visible` is `true`, and that `pointStyle` and `pointRadius` (or `pointSize`) are set appropriately.

Once created, markers can be removed by selecting them and choosing `Delete` from the context menu.

## 6 Editing Properties

Most ArtiSynth model components have properties which can be changed onscreen through the graphical interface. Properties include a diverse set of attributes ranging from stiffness and damping for `AxialSprings`, position and velocity for particles and rigid bodies, or whether or not a component is dynamic.

The underlying software architecture of the property interface is described in the Properties chapter of the [Maspack Reference Manual](#).

### 6.1 Property panels

To edit properties for a set of components, select the components in question, then right-click in either the viewer or the navigation panel, and select `Edit properties`. This will create a *property panel* for all properties which are common to the selected components. All typical property panel is shown in Figure 26.

Property panels are initialized with the current values of the selected components, providing a view of the current property state. A blank property value will be displayed when more than one component is selected and the corresponding property value differs across components.

Some properties are *read only*. In this case, the corresponding widget in the property panel will display the value but will be disabled.

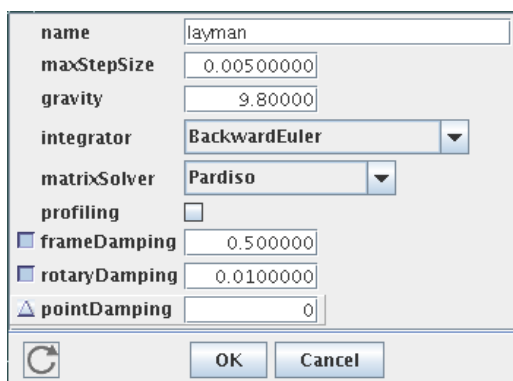


Figure 26: A typical property panel.

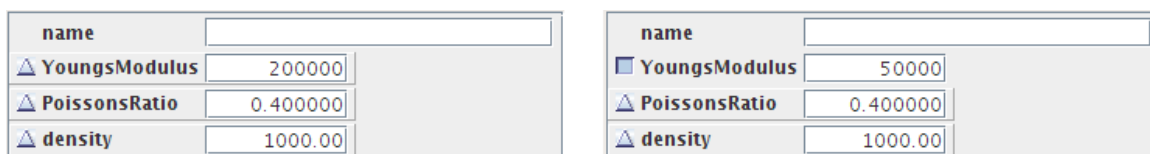



Figure 27: Property panel showing YoungsModulus as inherited (left) and explicitly set (right).

Property panels are non-modal and persistent. They can be deleted by closing them or clicking the OK button. Clicking the Cancel button will cause the properties to be reset to their values at the time the panel was created.

Normally, a property panel will refresh its widget values whenever the model view is rerendered. In particular, this will happen repeatedly while simulation is running. To disable the automatic refresh, click the live updating button  at the lower left of the panel.

### 6.1.1 Inheritable properties

Some properties are inheritable. The value of an inheritable property can be *explicitly* set or it can be inherited. If inherited, then it inherits its value from ancestor components further up the hierarchy. More specifically, if a property's value is inherited, then the value is obtained from the nearest ancestor in which the same property exists and is explicitly set. If no such ancestor exists, then the property is set to a default value.

The inherited/explicit status of an inheritable property is controlled by an additional button placed at the left of the property widget (Figure 27). Clicking this button toggles the inherited/explicit status. If set to inherited, then the property's value is determined from the hierarchy and the updated value is placed in the widget. Setting the value in the widget itself will cause the inheritable status to be set to explicit, and the value of inherited instances of the same property in descendant nodes will be updated accordingly.

## 6.2 Render properties

Render properties are associated with any component that is renderable. They are defined in the class [RenderProps](#). Because of their complexity, they are adjusted through a separate panel from the standard property panel.

To adjust the property panel for a set of components, select the components in question, using either the viewer window or navigation panel, and then right click in either the viewer or the navigation panel. Several options may appear in the context menu:

### Edit render props

This will create a special property panel allowing the render properties for the selected components to be set (see Section 6.2.1 and Figure 28).

### Clear render props

This will actually remove the render properties from the selected components (i.e., their render properties will be set to `null`). Nominally, this means that the components will not be rendered, *unless* their parents take responsibility for rendering children without render properties. The latter behavior is common for lists of particles, springs, finite elements, etc., in order to avoid the need for defining render properties in a large number of objects.

### Set visible

This option will appear if any of the selected objects are invisible. Selecting it will set the render properties so that all components are visible.

### Set invisible

This option will appear if any of the selected objects are visible. Selecting it will set the render properties so that all components are invisible.

## 6.2.1 Render property settings

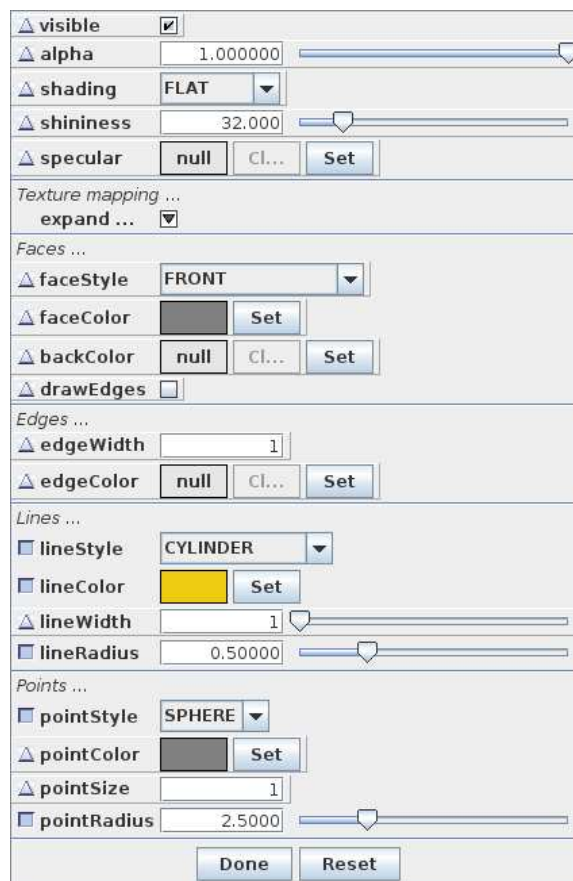


Figure 28: A typical panel for adjusting render properties.

There are a large number of render property settings. Loosely speaking, they are divided into generic settings, along with those related to faces, lines, and points. How these are used depends on what is being rendered. Mesh rendering typically uses the face settings, along with the line settings to render edges if the `drawEdges` property is set `true`. Line settings are also used for rendering axial springs and the edges of FEM elements. Point settings are used for rendering any subclass of `Point`, including `Particle` and `FemNode`.

Not all render properties may appear in a render panel; usually, only those properties relevant to the selected components will be presented.

### Generic properties:

**visible:** Whether or not the component is visible.

**alpha:** The transparency for polygonal faces (range 0 to 1. Default is 1, for opaque).

**shading:** How polygons are shaded (FLAT, SMOOTH, METAL and NONE). For viewer implementations there may be no difference between SMOOTH and METAL.

**shininess:** Shininess parameter for polygons (range 0 to 128). Default is 32.

**specular:** If not null, specifies the specular reflectance color.

#### Face related properties:

**faceStyle:** Which polygonal faces are drawn (FRONT, BACK, FRONT\_AND\_BACK, NONE).

**faceColor:** Color used for drawing faces.

**backColor:** Color used for drawing backs of faces. If null, faceColor is used.

**drawEdges:** If true, face edges of the polygons are drawn explicitly.

#### Texture mapping properties:

**colorMap:** If not null, specifies the image source file and properties for color mapping.

**normalMap:** If not null, specifies the image source file and properties for normal mapping.

**bumpMap:** If not null, specifies the image source file and properties for bump mapping.

#### Edge related properties:

**edgeColor:** The color for edges.

**edgeWidth:** Edge width in pixels.

#### Line related properties:

**lineStyle:** How lines are drawn (CYLINDER, LINE, or SPINDLE).

**lineColor:** The color for lines.

**lineWidth:** Line width in pixels when LINE style is selected.

**lineRadius:** Cylinder radius when CYLINDER or SPINDLE style is selected.

#### Point related properties:

**pointStyle:** How points are drawn (SPHERE or POINT).

**pointColor:** The color for points.

**pointSize:** Point size in pixels when POINT style is selected.

**pointRadius:** Sphere radius used when SPHERE style is selected.

A typical panel for editing render properties is shown in Figure 28. Texture mapping properties, if present, are normally hidden by default and can be exposed by clicking on the expand... button.

## 7 The Timeline

The *timeline* is a panel that provides “play” controls for starting and stopping the simulation, and allows the user to graphically arrange temporal sequences of probes and waypoints to control and monitor the simulation. If not initially visible, its visibility can be toggled by hitting the ‘t’ key from within the viewer (Section 3.11).

## 7.1 Probes and waypoints

ArtiSynth allows models to connect to special components, known as *probes*, which can either supply input values or monitor output values over time as the simulation proceeds. Probes attached to simulation inputs are known as *input probes* (class [InputProbe](#)), while those attached to outputs are known as *output probes* (class [OutputProbe](#)). Each probe has a *start time* and a *stop time*, and implements an [apply\(t0,t1\)](#) method that supplies (or collects) data for the simulation step between time  $t_0$  and  $t_1$ . Probes are described in more detail, along with specifics about how to code them into applications, in the “Simulation Control” chapter of the [ArtiSynth Modeling Guide](#).

The most commonly used probe subclasses are [NumericInputProbe](#) and [NumericOutputProbe](#), each of which is associated with a vector of floating point values that are interpolated over time. This data is usually connected to various model component properties, and used to either set (for input probes) or collect (for output probes) the values of those properties. The size of the vector is known as the probe’s *vector size* and matches the properties that the probe is connected to. For example, a probe controlling a single muscle excitation value will have a vector size of one, whereas a probe collecting the 3D velocity of a point will have a vector size of three.

Within a numeric probe, data is defined by a temporal sequence of *knots points* which give the vector values at pre-scribed times, with values in between determined by interpolation (Section 7.5.7). For input probes, the knot density is often sparse, whereas for output probes it matches the sample rate at which data is collected, which is usually the simulation step size.

Input and output probes are arranged and displayed graphically in the timeline, within a set of *tracks* (Section 7.4). Each probe is displayed as a bar within one of these tracks. The display for numeric probes can also be expanded to show a graph of the numeric data (Section 7.5).

ArtiSynth allows models to set temporal *waypoints*, which are designated times at which the model state is saved during simulation. This allows the simulation to be later reset to that time without having to recompute the simulation from the beginning. A special type of waypoint is known as a *breakpoint*, which causes the simulation to pause when it is reached. The timeline displays the waypoints, and allows them to be created and edited (Section 7.3).

## 7.2 Basic timeline structure

The basic structure of the timeline is shown in Figure 29.

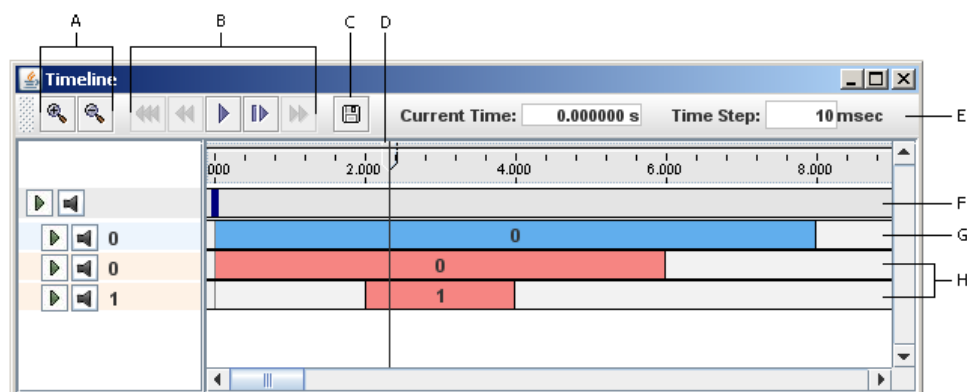


Figure 29: The timeline panel, containing: (A) zoom controls, (B) play controls, (C) save button, (D) time cursor, (E) toolbar, (F) waypoint track, (G) input track, (H) output track.

The *toolbar* at the top contains the following widgets:

### Zoom controls

Shrinks or expands the timescale.

### Play controls

Starts, pauses, or resets the simulation.

### Save button

Saves the data for all output probes with attached files.

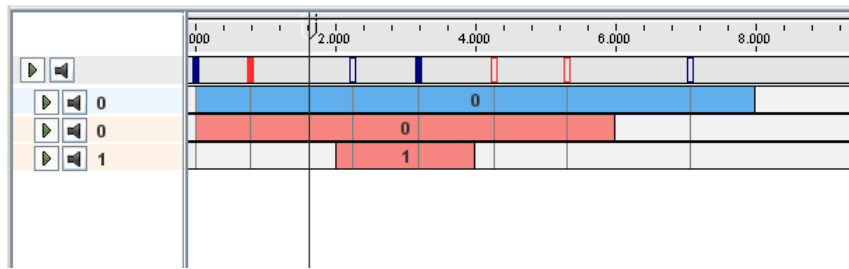


Figure 30: Waypoints (blue) and breakpoints (red) arranged in the waypoint track.

### Time box

Shows the current simulation time.

### Time step box

Shows the length of time associated with a “single step” command.

## 7.2.1 Play controls

The play controls are in turn comprised of the following buttons:



Reset: Resets the simulation to the beginning at time 0.



Skip-back: Moves the simulation time backward to the previous valid waypoint (see Section 7.3).



Play: Starts the simulation.



Pause: Takes the place of the play button and pauses the simulation.



Single-step: Advances the simulation by a single step, specified in the time step box.



Skip-forward: Moves the simulation time forward to the next valid waypoint (see Section 7.3).



Stop-all: Stops the simulation *and* any currently running Jython commands and scripts (Section 12.3).

## 7.2.2 Tracks

The timeline proper is divided into a number of tracks. At the top is the *waypoint track*, which is used to arrange waypoints and breakpoints. Below that are a variable number of *input* and *output* tracks, which are used respectively for arranging the input and output probes.

## 7.3 Viewing and setting waypoints

### 7.3.1 Waypoints

Waypoints are arranged along the waypoint track at the top of the timeline and are indicated by a small rectangular blue box (Figure 30). A solid box indicates a waypoint that contains a valid state and thus can be advanced to using the fast forward/backward buttons.

To add a waypoint, right-click on the waypoint track. A popup menu will show a number of options, including **Add waypoint here**, which adds a waypoint at the current location of the time cursor, and **Add waypoint(s)**, which will bring up a dialog prompting for a specific time to add a Waypoint. The "Add Waypoint" dialog also contains a repeat field,

which will cause additional waypoints to be added with a spacing indicated by the time value, and an option to create breakpoints instead of waypoints.

Once created, waypoints can be moved by dragging them. They can also be deleted by right-clicking on them and selecting the Delete waypoint option.

To delete all the waypoints, select Delete waypoints, either from the main File menu, or after right-clicking on the waypoint track.

### 7.3.2 Breakpoints

Breakpoints are waypoints that also cause the simulation to stop. They are displayed in red instead of blue.

Breakpoints can be added in the same way as waypoints, i.e., by right clicking on the waypoint track and selecting Add breakpoint here or Add waypoint(s).

Waypoints can be converted into breakpoints (and vice versa) through the context menu.

### 7.3.3 Saving and loading

It is possible to save and load waypoints to and from an external file. The following menu options may be selected to do this, either from the main File menu, or after right-clicking on the waypoint track:

#### Save waypoints as ...

Brings up a file chooser that allows the user to specify a file for saving all waypoints and their state data. Clicking Save As completes the operation.

#### Save waypoints

If a waypoint file has already been specified using either “Save waypoints as ...” or “Load waypoints ...”, then all waypoints and their state data are saved to that file.

#### Load waypoints ...

Brings up a file chooser that allows the user to specify a file from which waypoints and their state data will be loaded. Clicking Load completes the operation. The new waypoints are added to any existing ones, but previously waypoints are not deleted. Checks are made to help ensure that the waypoint data is consistent with the model’s current state.

#### Reload waypoints

Identical to “Load waypoints ...”, except that it uses a file that has already been specified using either “Save waypoints as ...” or “Load waypoints ...”.

#### Delete waypoints

Deletes all waypoints *except* the one at time zero.

Waypoint files are currently stored as binary files. The reason for this is that the required storage is about 1/2 of that required for text files, while the writing and parsing times are as much as 10× faster.

## 7.4 Tracks and probes

Probes are arranged on tracks located beneath the waypoint track. Input probes must be placed on input tracks and output probes must be placed on output tracks. Furthermore, probes on the same track are not allowed to overlap.

#### Note:

In the future, additional restrictions may be placed on what type of probe can be placed on what track.

Probes can be moved horizontally to different times as well as vertically onto different tracks. They can also be stretched by dragging the edges of the probe and cropped by holding the control key while stretching.

On the left side of the timeline is the track panel, which contains a number of track control widgets (Figure 31).

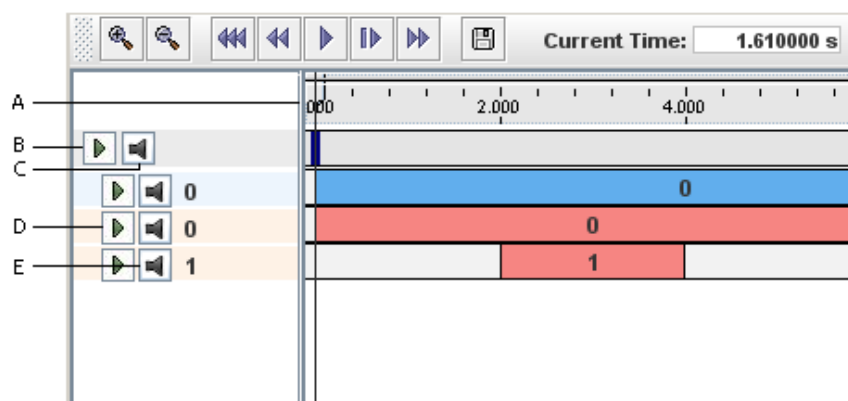


Figure 31: Close up of the track panel, showing: (A) track panel, (B) expand all button, (C) mute all button, (D) expand button, (E) mute button.

#### 7.4.1 Creating, moving, and deleting tracks

New tracks may be added by right-clicking in the waypoint track and selecting **Add input track** or **Add output track**.

The vertical location of a track can be moved by left clicking on it in the left panel and dragging it up or down to a new location.

A track can be deleted by right-clicking on the track in the track panel and selecting **Delete track**.

#### 7.4.2 Muting tracks

A track can be *muted* or *unmuted* by clicking on its gray mute button in the track panel (Figure 31). All probes on a muted track are ignored during simulation.

All tracks can be muted, thus disabling all probes, by clicking on the mute all button in the gray panel above the tracks.

#### 7.4.3 Expanding tracks

A track can be *expanded* or *collapsed* by clicking on its green expand button in the track panel (Figure 31). Expanding a track creates an additional area in which the data associated with the track's probes may be displayed (see Figure 32). The way in which this data is displayed is probe-specific. Probes containing numeric data usually show their data graphically, as described in Section 7.5.

#### 7.4.4 Grouping tracks

Multiple contiguous tracks can be selected by clicking on them while holding the control key. Furthermore, they can be *grouped* together or *ungrouped* by selecting the appropriate option from the context menu. Grouped tracks can be collapsed, moved simultaneously and muted together.

### 7.5 Numeric probe displays

Data associated with numeric probes is displayed as a graph within the display (Figure 32), with each entry in a probe's data vector drawn as a separate trace. If the probe's vector size is greater than one, each trace is drawn using a different color (up to a limit, after which colors are reused).

#### 7.5.1 Setting the range and display properties

The range and other properties of the display can be set by right clicking in the display and selecting **"Edit range and properties ..."**, which creates a dialog that allows these to be adjusted. If the `autoRanging` property is set to `true`, then the

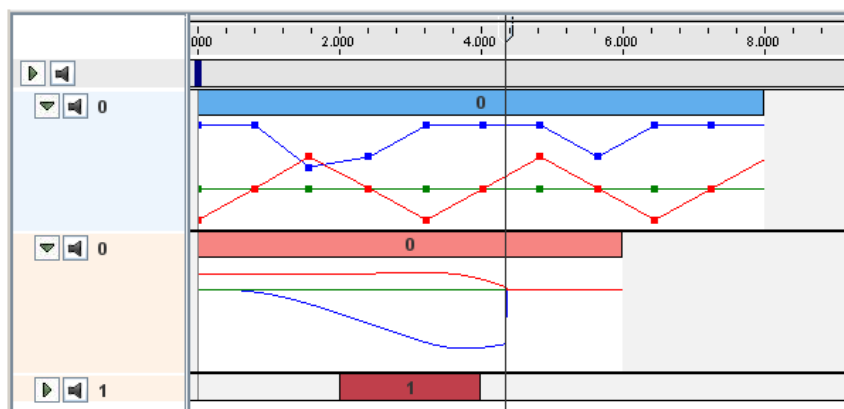


Figure 32: Expanded input and output tracks showing numeric data.

display range automatically expands as needed to accommodate new data. Display ranges can also be adjusted to fit the current data by right-clicking in the display and selecting Fit ranges to data.

Large data displays (Section 7.5.2) contain additional GUI-based features to adjust the display range.

### 7.5.2 Large displays

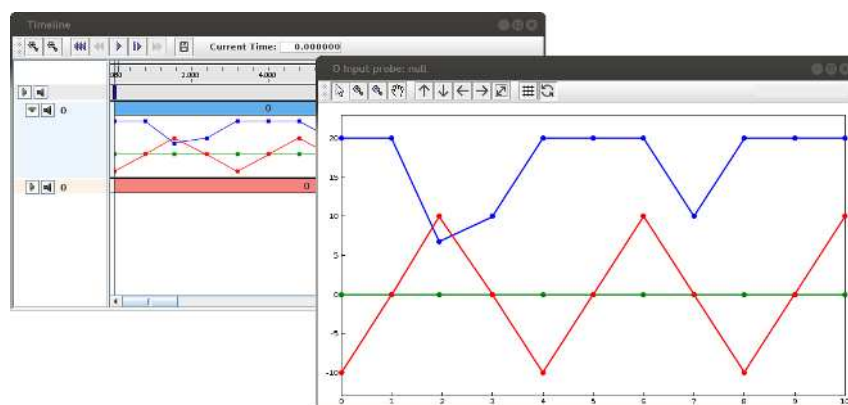









Figure 33: A large display for a numeric probe.

A large display for a numeric probe can be created by right-clicking on the probe icon and selecting Large Display. This will create a large numeric display in a separate panel, allowing more precise inspection of data (Figure 33).

In addition to the functionality of the smaller displays, large displays have additional buttons, located across the top, for controlling the display range and other items. The first four of these are mode buttons:

- |  |                   |   |
|--|-------------------|---|
|  | <b>Select:</b>    | Places the display into <i>selection</i> mode, allowing knots (when visible) to be edited as described in Section 7.5.5.  |
|  | <b>Zoom in:</b>   | Places the display into <i>zoom in</i> mode, in which the user can zoom in by either drag-selecting a region, or by left clicking on a point of interest. The mouse wheel can also be used to zoom in or out. |
|  | <b>Zoom out:</b>  | Places the display into <i>zoom out</i> mode, in which the user can zoom out by left clicking on a point of interest. The mouse wheel can be also used to zoom in or out.                                     |
|  | <b>Translate:</b> | Places the display into <i>translate</i> mode, in which the user can translate a (zoomed-in) display using the left mouse button. The mouse wheel can also be used to zoom in or out.                         |

There are also several additional buttons:

	Increase y:	Increases the vertical y range.
	Decrease y:	Decreases the vertical y range.
	Decrease x:	Decreases the horizontal x range.
	Increase x:	Increases the horizontal x range.
	Fit Range:	Fits the vertical and horizontal ranges to the current data.
	Grid:	Enables or disables visibility of a grid that aligns with the x and y axis tick marks.
	Auto range:	Enables or disables <i>auto-ranging</i> , in which the y axis is automatically adjusted to fit new data.

### 7.5.3 Cloning displays and exporting plots

Large data displays can be *cloned* by right-clicking in the display and selecting Clone display. This creates a duplicate display (Figure 34) containing a copy of all the data currently in the probe. However, the duplicate display is not attached to the probe, and so the data does not change when the probe is reset or additional data is added to the probe. This is useful for comparing outputs between different simulations.

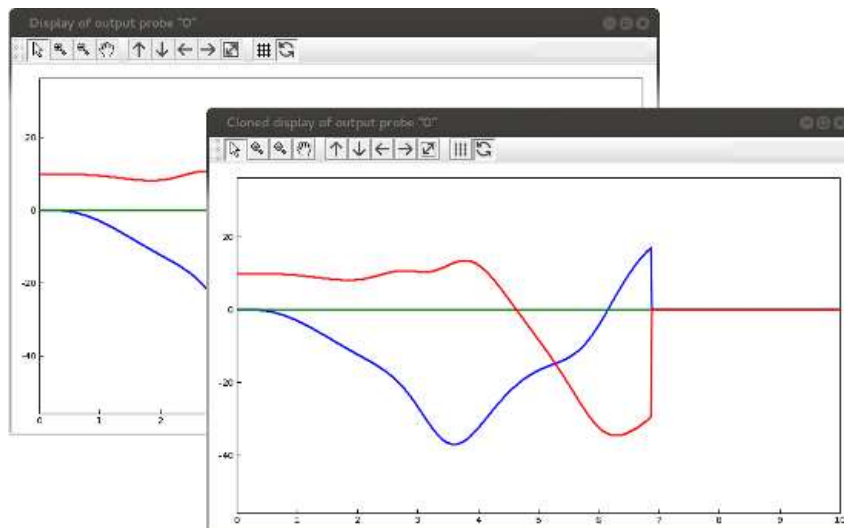


Figure 34: A large display of an output probe and its clone.

In addition, it is possible to export a large display's plot to an image file. Right-click in the display, choose "Export image as ...", and enter the desired file name and file type in the chooser. The file's type is indicated by its extension. A range of image file types are supported, including JPEG (.jpg, .jpeg), PNG (.png), scalable vector graphics (.svg), and encapsulated PostScript (.eps).

### 7.5.4 Legends and visibility control

As mentioned above, each entry in a numeric probe's data vector is rendered in a different color (up to a limit, currently 30, after which colors are reused). If the vector size is large (say more than three or four), or if there is much overlapping of values, then the result can be difficult to visualize.

To manage this problem, numeric displays provide a *legend* dialog that allows the user to control the color, drawing order, and visibility for each vector entry (Figure 35).

To open the legend dialog, right-click in the display panel and select Show legend. Each row in the legend dialog is associated with a specific trace, consisting of a descriptive label followed by the trace color and a toggle controlling its visibility. For traces associated with component properties, the label consists of a partial path identifying the property, and for vector-valued properties this is followed by the name or number of the corresponding vector element. Traces are rendered in bottom-to-top order, so those at the top will be most visible.



Figure 35: Legend dialog for the displays of Figure 33. The label for each entry shows its associated property and vector element value (in this case, the  $x$ ,  $y$  and  $z$  values for the `targetPosition` property of the particle `pnt0`).

- To change a trace's color, click the **Set** button, which will bring up a color menu.
- To make an trace visible or invisible, use the **visible** toggle box.
- To make *all* traces visible or invisible, use the **All visible** toggle box at the bottom of the dialog.
- To change the order in which a trace is drawn, click and drag its dialog entry vertically.

Traces can also be reordered using the **Trace ordering** selector at the dialog bottom. At the time of this writing, the available orderings include:

### ORDINAL

Traces are ordered by their natural place in the probe data. This is the default.

### RMS

Traces are ordered in descending order by the RMS (root mean square) of their values over time.

### RANGE

Traces are ordered in decreasing order by the difference between their maximum and minimum values over time.

Reordering traces, and then restricting visibility to a few at the top, can allow visualization of the most prominent data in an otherwise crowded display. For example, Figure 36 shows the 16 muscle excitations produced by inverse simulation of a muscular hydrostat model, with the legend dialog shown in Figure 37 (left). To view only the five excitations with the largest RMS values, one can use the **Trace ordering** selector to sort them in decreasing RMS order, and then make all traces invisible except the top five. The resulting display is shown in Figure 38, with the corresponding legend dialog shown in Figure 37 (middle).

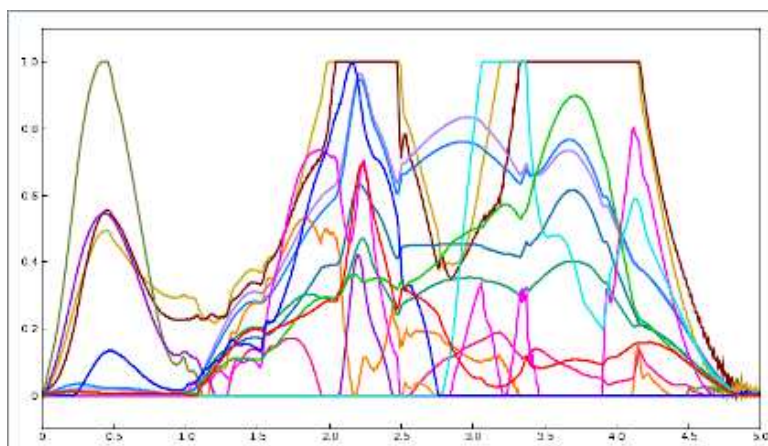


Figure 36: Display showing the muscle excitations produced by inverse simulation of a muscular hydrostat.

When traces are reordered, their colors are not changed. To reassign the default colors to the traces based on their current ordering, one may use the **Reset Colors** button at the bottom of the legend dialog. Figure 39 shows the result of doing this for the display of Figure 38, with the corresponding legend dialog shown in Figure 37 (right).

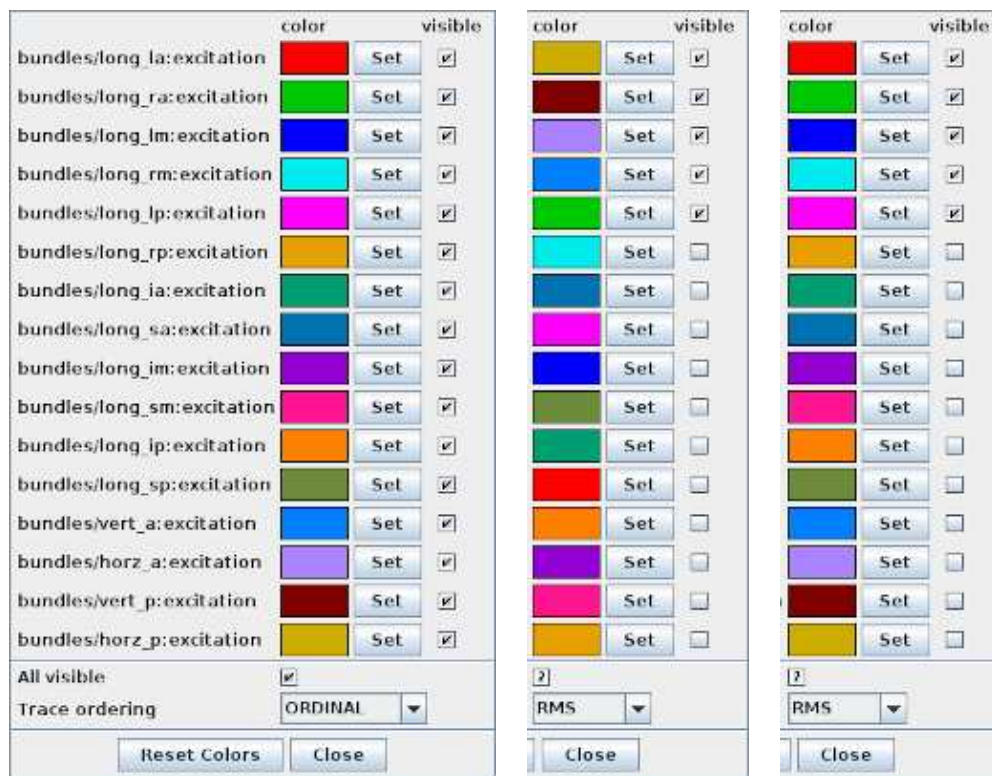


Figure 37: Left: legend dialog for Figure 36. Middle: legend dialog for Figure 38, where the traces have been sorted in decreasing RMS order and only the first five traces have been made visible. Right: legend dialog for Figure 39, where the “Reset Colors” button has been used to reset the trace colors to their canonical values with respect to the current ordering.

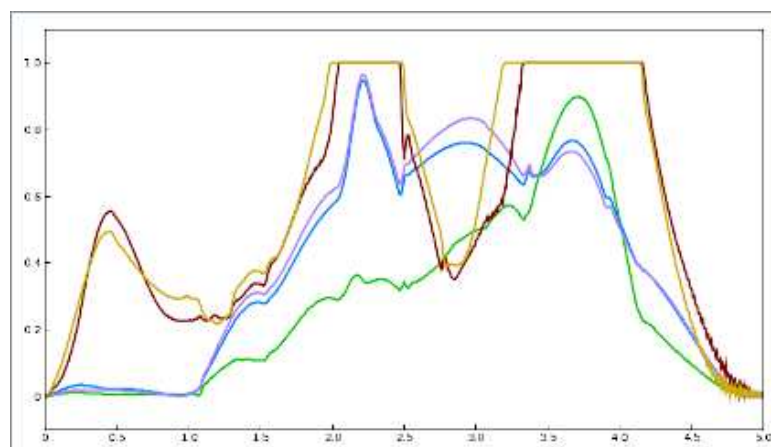


Figure 38: Display showing the five excitations from Figure 36 with the largest RMS, arranged by sorting the traces by maximum RMS and then making only the first five visible.

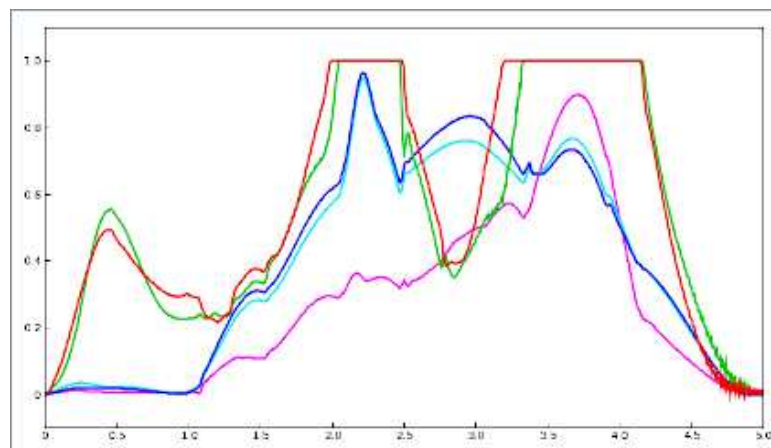


Figure 39: Display showing the excitations from Figure 38, with the trace colors reset to their canonical values with respect to the current ordering.

### 7.5.5 Editing and scaling data

As mentioned in Section 7.1, numeric probe data is described by a temporal sequence of *knots*, between which data is interpolated as described in Section 7.5.7.

Knot points can be made visible or invisible by setting the display's `knotsVisible` property. This can also be done by right-clicking in the display and selecting Show knot points or Hide knot points. The rendered size of the knot, in pixels, is controlled by the `knotSize` property.

Since output probes typically have a very high knot density, their knots are set to be invisible by default.

For input probes, knot points (when visible) can be edited by moving, adding, or deleting them:

- To move a knot point, left click on the knot and drag it.
- To add a knot point, place the mouse at the desired location and double left-click.
- To delete a knot point, right-click on the knot and select Delete knot point.
- To edit a knot point data value, right-click on the knot and select Edit knot point.

The data for all numeric probes (input or output) can be scaled by right-clicking in the display and selecting “Scale data ...”. This allows the user to enter a scaling factor that is applied uniformly to all the knot points.

### 7.5.6 Smoothing data

Numeric probe data can also be smoothed, which is convenient for removing noise from either input or output data. To apply data smoothing, right click in the display and select “Smooth data ...”; this will open a *smoothing dialog* as shown in Figure 40.

Figure 40: Dialog for smoothing probe data.

Different smoothing methods are available and can be selected using the dialog's method field. At the time of this writing, the following methods are available:

#### MOVING\_AVERAGE

Applies a mean average filter across the knots, using a moving window whose size is specified by the window size field. The window is centered on each knot, and is reduced in size near the end knots to ensure a symmetric fit. The end knot values are not changed. The window size must be odd and the window size field enforces this.

#### SAVITZKY\_GOLAY

Applies Savitzky-Golay smoothing across the knots, using a moving window of size  $w$ . Savitzky-Golay smoothing works by fitting the data values in the window to a polynomial of a specified degree  $d$ , and using this to recompute the value in the middle of the window. The polynomial is also used to interpolate the first and last  $w/2$  values, since it is not possible to center the window on these.

The window size  $w$  and the polynomial degree  $d$  are specified by the window size and polynomial degree fields.  $w$  must be odd, and must also be larger than  $d$ , and the fields enforce these constraints.

Note that smoothing makes no adjustment to account for uneven knot spacing, which may sometimes occur for input data.

### 7.5.7 Interpolation control

The data between knot points in a numeric probe is interpolated, using one of the interpolation *orders* described below, with the default interpolation order being Linear. Linear interpolation is almost always sufficient for output probes, which typically have a close spacing between knots that equals the simulation step size. Input probes, on the other hand, often have a much sparser knot spacing and so different interpolation orders can be useful. The interpolation order of input probes can be set by right-clicking in the display and selecting the Interpolation menu item. This is illustrated in Figure 41, which shows two input probes, one using Cubic interpolation and the other using Linear interpolation.

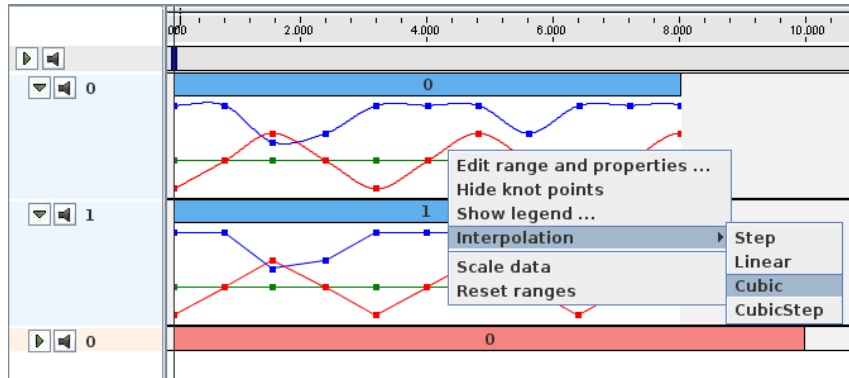


Figure 41: Two expanding tracks with probes showing cubic interpolation (upper) and linear interpolation (lower).

The interpolation options for a numeric probe include:

#### Step

Values are set to the values of the closest previous knots points.

#### Linear

Values are set by linear interpolation of the closest surrounding knot points.

#### Cubic

Values are set by cubic Catmull interpolation between the surrounding knot points.

#### CubicStep

Values are set by cubic Hermite interpolation between the surrounding knot points, with the slopes at knot points set to zero.

#### SphericalLinear

When interpolating quaternions or  $4 \times 4$  rigid transformation matrices, 3D rotation values are set by piecewise spherical linear interpolation (i.e., "slerp", as described by Shoemake's 1985 SIGGRAPH paper). Otherwise, interpolation is linear. Quaternions are assumed if the vector size of the numeric probe is 4, and  $4 \times 4$  rigid transformation matrices are assumed if the vector size is 16.

#### SphericalCubic

When interpolating quaternions or  $4 \times 4$  rigid transformation matrices, 3D rotation values are set by spherical cubic interpolation (i.e., "slerp", as described by Shoemake's 1985 SIGGRAPH paper). Otherwise, interpolation is cubic. Quaternions are assumed if the vector size of the numeric probe is 4, and  $4 \times 4$  rigid transformation matrices are assumed if the vector size is 16.

All of the above interpolation orders are instances of the enumeration type [Interpolation.Order](#).

## 8 Saving and Loading Probes

### 8.1 Saving and loading probe data

Each ArtiSynth probe can be associated with an *attached file* to (or from) which its data can be saved (or loaded). The attached file is specified by the probe's `attachedFile` property, which is a string giving the file's path name. An absolute

path locates the file relative to the system root folder, while a relative path locates it relative to the current value of ArtiSynth's working folder (Section 2.10). If the `attachedFile` property is `null`, then the probe does not have an attached file.

For numeric probes, the file format used to save and load data is described in the “Data file format” subsection of the “Simulation Control” chapter of the [ArtiSynth Modeling Guide](#).

Data for an individual probe can be saved or loaded by first selecting it (either within the navigation panel or by left-clicking on its timeline display bar), and then right-clicking and choosing one of the following options from the resulting pull-down menu:

#### Save data

If the probe has an attached file, saves its data to that file.

#### Save data as ...

Brings up a file chooser allowing the user to specify an attached file for the probe. Clicking the Save As button then sets the attached file and saves the probe's data in it. The probe's `attachedFile` property will be set to a relative path name if the file is located beneath the current ArtiSynth working folder (Section 2.10), and an absolute path name otherwise.

#### Load data

If the probe has an attached file, loads its data from that file.

#### Load data from ...

Brings up a file chooser allowing the user to specify an attached file for the probe. Clicking the Load button then sets the attached file and loads the probe's data from it. The probe's `attachedFile` property will be set to a relative path name if the file is located beneath the current ArtiSynth working folder, and an absolute path name otherwise.

In addition, data can be saved for *all* output probes that have attached files by either selecting “Save output probe data” from the ArtiSynth File menu, or by clicking on the timeline's save button (C in Figure 29).

## 8.2 Exporting numeric probe data

The data associated with a numeric probe can also be exported to either a CSV file (\*.csv) or a regular text file (\*.txt).

Each line in the file describes the numeric data associated with one of the probe's knot points, and consists simply of a sequence of  $n$  numbers, where  $n$  is the probe's vector size. For CSV files, the numbers are separated by commas, while for text files they are separated by spaces. If “include time data” is selected in the export dialog (see below), then the knot's time value is also included at the beginning of the line, so that the complete sequence includes  $n + 1$  numbers.

The time values exported with a probe are *probe relative*, so that  $t = 0$  corresponds to the probe's start time and any scaling is ignored.

To export a probe's data, first select the probe (either within the navigation panel or by left-clicking on its timeline display bar), and then right-click and choose “Export data as ...” from the resulting pull-down menu.

This brings up a dialog (Figure 42) that allows the user to specify the file name and output file type (CSV or text). It also allows the user to specify export-specific properties, such as:

#### numeric format

A printf-style format string specifying the formatting for floating point data. The default value is `%g`, which means all numbers will be written to full precision with a variable length and using scientific notation if needed. Other allowed formats are described in the documentation for [maspack.util.NumberFormat](#).

#### include time data

A Boolean value which if `true` indicates that time data should be included along with the numeric values. Not including time data may be appropriate when the data is spaced at a known time interval.

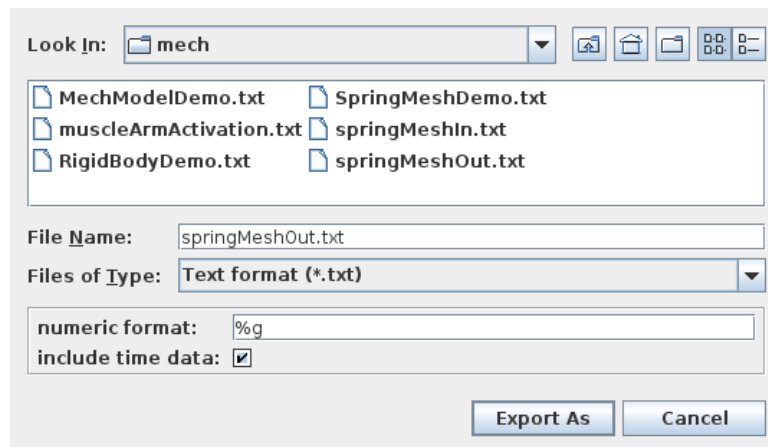


Figure 42: The export probe data dialog.

### 8.3 Saving and loading all probes

The model's entire configuration of input and output probes can be saved to (or loaded from) a single file. This file stores, in an ASCII text format, complete information about each probe, including its start and stop times, location within the timeline, which model properties it connects to, the name of its attached file (if any), and its current data. The file also stores the current waypoint locations, but does *not* store waypoint data.

Once saved, a probe file can be later loaded to reset the entire probe configuration.

Probes can be saved or loaded by either opening the ArtiSynth File menu, or by right-clicking on the timeline's waypoint track, and selecting one of the following:

#### Save probes as ...

Brings up a file chooser that allows the user to specify a file for saving the current probe configuration. Clicking Save As completes the operation.

#### Save probes

If a probe file has already been specified using either "Save probes as ..." or "Load probes ...", then the current probe configuration is saved to that file.

#### Load probes ...

Brings up a file chooser that allows the user to specify a file from which the probe configuration will be loaded. Clicking Load completes the operation.

## 9 Adding and Editing Numeric Probes

Numeric input and output probes can be interactively added to a simulation. Output probes allow you to record a vector of values derived from one or more model component properties. Input probes allow you to use a vector of input data to drive one or more model component properties.

### 9.1 Adding output probes

To add a numeric output probe, go to the main menu and choose `edit > add output probe`. This will create a numeric output probe editor, as shown in Figure 43.

The editor contains three main panels:

- A *property panel* at the upper left in which allows you to add or edit the properties of model components whose values will be used in computing the final output probe value. Each property is associated with a component/property widget, which allows you to first select a component and then choose one of its properties.

Figure 43: Probe editor for an uninitialized numeric output probe.

- A *formula panel* at the upper right which allows you to add or edit formulae which convert the values from the properties into numeric values for the output probe.
- A *probe property panel* at the bottom which allows properties of the probe itself to be set.

### 9.1.1 Creating a simple probe

Creating an output probe that simply records the value of a single numeric property is fairly easy. Starting with the output probe editor in Figure 43:

1. Select a component, either externally through the navigation panel (Section 4.2), the viewer (Section 4.3), or the selection display (Section 4.4), or by entering its path name into the component/property widget. Once a component is selected, the left-most “up” arrow can be used to select that component’s parent.
2. Select a property for the component from the property selection box at the right of the component/property widget.
3. Adjust any required properties for the probe itself (Section 9.3).
4. Click Done.

### 9.1.2 General output probes

The incorporation of Jython formulae into output probes as described in this section can be difficult to implement in code. Instead, users are encouraged to use [NumericMonitorProbes](#) to create output probe data based on general functions. See the section “Numeric monitor probes” in the [ArtiSynth Modeling Guide](#).

In general, output probes define a general map from the numeric values of  $n$  model component properties to a generalized output vector formed by the concatenation of  $m$ -subvectors formed by  $m$  formulae, as shown in Figure 44. In the simple case of Section 9.1.1, there is a single property, one sub-vector equal to the output vector, and a formula which is just the property value itself.

Each of the  $n$  properties has a numeric value which is represented by a variable  $p_i$  and which is a vector of some dimension (scalars being considered vectors of dimension 1). The dimension of the  $p_i$  depends on the property and is displayed to the right of the property selection box on the component/property widget.

Each of the  $m$  formulae is an arithmetic expression, implemented in Jython (Section 12), which may involve one or more of the variables  $p_i$ . The output from each formula is itself a vector of dimension  $d_j$ , which is displayed at the right of each formula panel. The simplest formula is just a single variable  $p_i$ , in which case  $d_j$  equals the dimension of  $p_i$ . The concatenation of all the output vectors from all the formulae produces the output vector associated with the probe, which has a dimension  $\sum_m d_j$ .

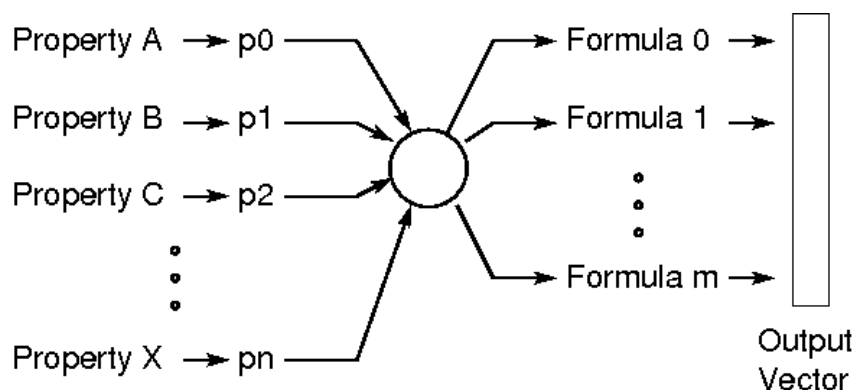


Figure 44: General mapping for an output probe.

### 9.1.3 Using the probe editor

What the probe editor allows you to do is create the above mentioned map by selecting the properties of model components, assigning variable names to them, and creating formulae using these variables. When all the selected components form a coherent mapping, the Done button will be enabled and the probe can be completed. When one or more parts of the mapping is unspecified or incomplete, the associated widgets will be highlighted and the Done button will be disabled.

Extra properties can be added by requesting additional component/property widgets using the "+" button in the property panel. Similarly, extra formulae can be requested using the add button in the formula panel.

Properties and formulas can also be deleted; simply right click on the associated widget and choose delete.

Property variable names appear in a box at the left of the component/property widget. Variable names can be changed by editing this box. Name changes will be propagated into the formulae.

In order to streamline the probe creation process, the editor will try to guess certain desired actions. In particular, when the user chooses a property with a given component/property widget for the first time, the editor assigns a variable name to that property and creates a formula panel containing that variable. The variable name and formula panel can be changed if necessary.

#### Note:

The selection manager is connected to at most one component/property widget at a time. The component field for this widget is indicated with a blue border; external selections will affect only that widget. Left clicking on a component/property widget will cause it to be connected to the selection manager.

## 9.2 Adding input probes

To add a numeric input probe, go to the main menu and choose edit > add input probe. This will create a numeric input probe editor, as shown in Figure 45.

The editor contains three main panels:

- An *input panel* at the upper left which allows you to add or edit input variables. Each of these variables represents a sub-vector of the probe's numeric input vector.
- A *formula/property panel* at the upper right which allows you to add or edit numeric properties (using component/property widgets), along with formulae to determine values for these properties based on the input variables.
- A *probe property panel* at the bottom which allows properties of the probe itself to be set.

### 9.2.1 Creating a simple probe

Creating an input probe that simply sets a single numeric property from the probe's input vector is fairly easy, and is exactly analogous to creating a simple output probe (Section 9.1.1). Starting with the input probe editor in Figure 45:

Name	Dim	Formula	Component	Property
			<input type="text" value="models/msmod/particles/4"/>	
<div> <div> name <input type="text"/></div> <div>start time <input type="text" value="0.0000"/></div> <div>end time <input type="text" value="5.0000"/></div> </div> <div> <div>scale <input type="text" value="1"/></div> <div>attached file <input type="text"/></div> <div></div> </div>				

Figure 45: Probe editor for an uninitialized numeric input probe.

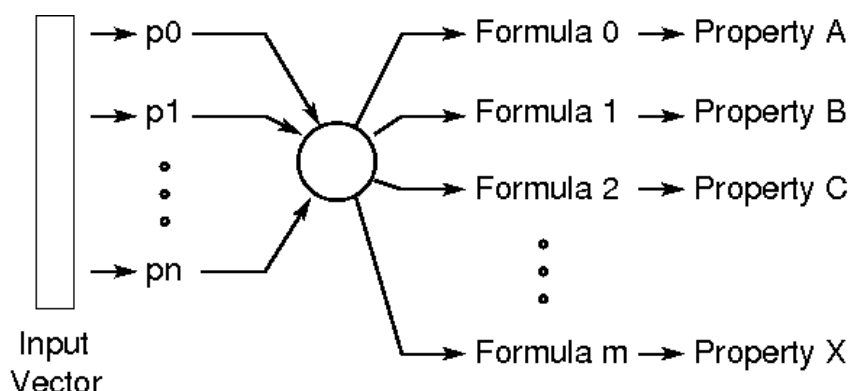


Figure 46: General mapping for an input probe.

1. Select a component, either externally, or using the component/property widget.
2. Select a property for the component from the property selection box at the right of the component/property widget.
3. Adjust any required properties for the probe itself (Section 9.3).
4. Click Done.

## 9.2.2 General input probes

The incorporation of Jython formulae into input probes as described in this section can be difficult to implement in code. Instead, users are encouraged to use [NumericControlProbes](#) to create general control inputs based on input probe data. See the section “Numeric input probes” in the [ArtiSynth Modeling Guide](#).

In general, input probes define a general map from the probe’s input vector (which is subdivided into  $n$  input variables of dimension  $d_i$ ) to the values of  $m$  properties, where each value is determined by an independent formula based on the input variables (Figure 46). In the simple case of Section 9.2.1, there is one input variable which equals the input vector, and it drives a single property using a formula which is just the value of the input vector.

Each input variable is a vector of dimension  $d_i$  (scalars being considered vectors of dimension 1), and the sum of all the  $d_i$  equals the dimension of the input vector.

Each of the formulae controlling the properties is an arithmetic expression, implemented in Jython (Section 12), which may involve one or more of the input variables  $p_i$ . The output from each formula is itself a vector whose dimension must match the associated numeric property. The simplest formula is just a single variable  $p_i$ , in which case its dimension equals the dimension of  $p_i$ .

### 9.2.3 Using the probe editor

The probe editor allows you to create the above mentioned map by creating input variables, selecting properties, and creating formulae to drive these properties. When all the selected components form a coherent mapping, the Done button will be enabled and the probe can be completed. When one or more parts of the mapping is unspecified or incomplete, the associated widgets will be highlighted and the Done button will be disabled.

Extra properties can be added by requesting additional component/property widgets using the "+" button in the formula/property panel. Similarly, extra input variables can be requested using the add button in the formula panel.

Properties and input variables can also be deleted; simply right click on the associated widget and choose delete.

Each input variable is associated with a widget containing two text fields, the left one defining the variable's name and the right one its dimension. The name or dimension can be changed by editing these fields. Changes will be propagated into the formulae; formulae that are found to be incompatible with the changes will be cleared.

In order to streamline the probe creation process, the editor will try to guess certain desired actions. In particular, when the user chooses a property with a given component/property widget for the first time, the editor creates an input variable whose dimension matches the property, and a simple formula consisting solely of the input variable. The input variable and formula can be changed if necessary.

**Note:**

The selection manager is connected to at most one component/property widget at a time. The component field for this widget is indicated with a blue border; external selections will affect only that widget. Left clicking on a component property widget will cause it to be connected to the selection manager.

## 9.3 Setting probe properties

The lower part of the probe editor contains a set of fields for editing various probe properties.

**name**

The name of the of the probe.

**start time**

Start time for the probe, in seconds.

**stop time**

Stop time for the probe, in seconds.

**scale**

Specifies the scale  $s$  for this probe, which relates the internal probe time  $t_p$  to the external simulation (or timeline) time  $t$ . If  $t_{\text{start}}$  is the time at which the probe starts on the timeline, then  $t = t_p s + t_{\text{start}}$ .

**attached file**

Names the *attached file* for this probe, used for storing the probe's data. See Section 8.1.

**display range**

Minimum and maximum range values used for the graphical display of the probe's data. If these values are left blank, then the range is computed automatically.

**update interval**

(Output probes only). A time interval, in seconds, specifying how often data should be output to the probe.

## 10 Point Tracing

Tracing can be enabled for individual points within an ArtiSynth model, allowing their paths to be recorded and visualized over the course of a simulation.

Within the GUI, tracing can be enabled by selecting one or more points and choosing “Enable tracing” from the context menu. This will create a *tracing probe* for each point, which is an output probe that stores the point’s position over time and uses this to render the path in the viewer. The default tracing probe has a start time of 0 and a duration of 10 seconds. To create tracing probes that have different start times or durations, or other property settings that differ from the default, one may instead choose “Add tracing probe”. This will open a *select tracing* dialog (Figure 47) that allows various properties of the tracing probe to be explicitly set.

Figure 47: Dialog to explicitly set properties for a new tracing probe.

Once created, tracing probes can be viewed and inspected within the timeline (Section 7). They can also be selected either by clicking on their traces within the viewer, or by clicking on their navigation panel entry under the root model’s `outputProbes` list.

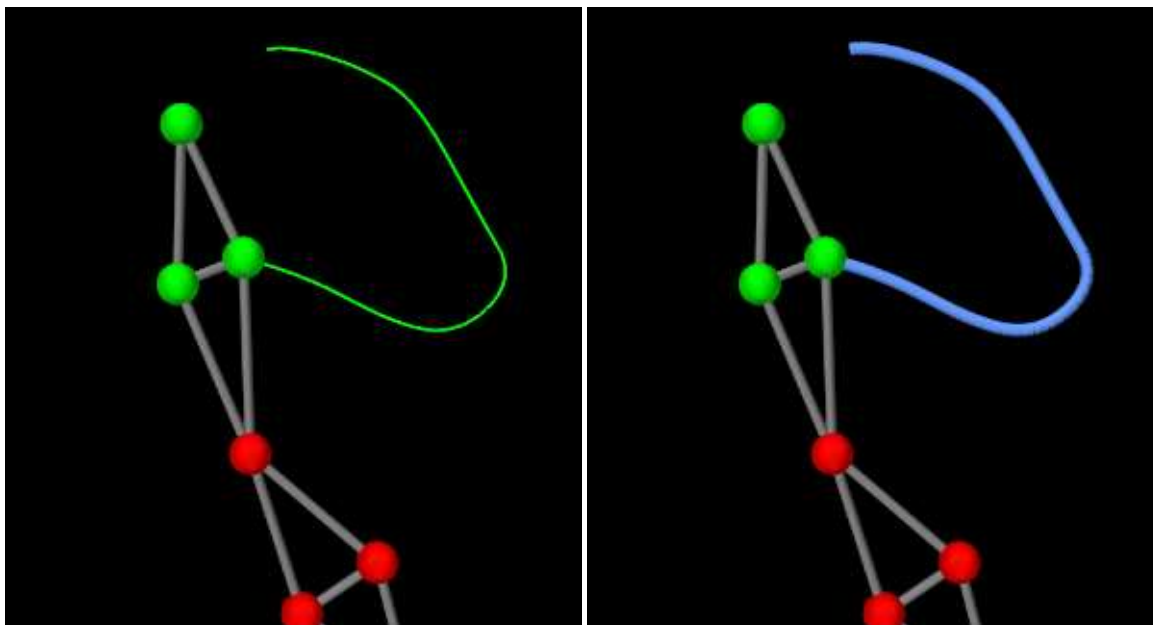


Figure 48: Point tracing rendered using the render properties for the tracing probe (left), and with `lineStyle`, `lineColor`, and `lineWidth` set to `CYLINDER`, pale blue and 0.6 (right).

A trace is rendered in the viewer as a curve formed from line segments between adjacent positions in the point’s path history. Its appearance can be controlled using the tracing probe’s render properties (Section 6.2.1), which can be adjusted by selecting the probe and choosing “Edit render props ...” in the context menu. Setting the `lineStyle`, `lineColor`, `lineWidth` and `lineRadius` properties will control the line style, color and width used to render the trace path. By default, trace curves are rendered using a `lineStyle` set to `LINE`, a `lineWidth` of 3 (pixels), and a `lineColor` set from the point’s

color. Figure 48 shows the demo `artisynt.demos.mech.SpringMeshDemo` after about 4 seconds of simulation with tracing enabled for point 4, using both the default and custom render properties set for the tracing probe.

Trace data is cleared when the simulation is restarted from time 0. It can also be cleared by selecting the corresponding point(s) and choosing “Clear trace” in the context menu.

Tracing can be removed from a point by deleting its tracing probe, which can be done either through the normal component deletion mechanism (selecting the probe and choosing Delete in the context menu), or by selecting the point(s) and choosing Disable tracing. The latter will remove all tracing probes associated with the point(s), including those created via “Add tracing probe”.

## 11 Settings and Preferences

A variety of settings are available to adjust attributes related to the viewer, mouse bindings, and model interaction and simulation. Many of these settings can also be saved as *preferences* (Section 11.2) to provide initial values when ArtiSynth is started up.

### 11.1 Settings

Settings for the viewer and mouse are discussed in Sections 3.9 and 3.10. Settings related to model interaction and simulation as discussed in the sections below.

#### 11.1.1 Interaction

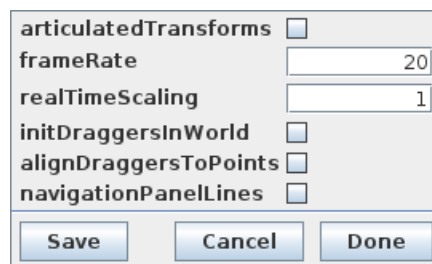


Figure 49: Interaction settings dialog.

*Interaction* settings involve attributes related to the graphical interface and model interaction. They can be adjusted by selecting “Interaction ...” from the Settings menu, which opens an *interaction settings dialog* (Figure 49). Clicking the Save button will save the current settings to the user’s preferences (Section 11.2) so that they will be set automatically when ArtiSynth is restarted. Interaction settings include:

##### **articulatedTransforms**

If `true`, attempts to preserve joint constraints when rigid bodies are moved using the manipulator controls (Section 5.2). The default value is `false`.

##### **frameRate**

Controls the rate, in frames per second, at which the viewer renders the simulation as it proceeds. The default value is 20.

If  $r$  denotes the frame rate, it is usually best to set  $r$  such that  $1/r$  is an integer multiple of the simulation step size. Otherwise, additional steps will be added to the simulation, at irregular intervals, in order to accommodate the requested rate.

##### **realTimeScaling**

This is a scaling factor for the viewer simulation speed.

**initDraggersInWorld**

If `true`, sets the initial orientation of transformer tools to be aligned with world coordinates, as described in Section 5.2. The default value is `false`.

**alignDraggersToPoints**

If `true`, and if `initDraggersInWorld` is `false`, then when a dragger is selected for a collection of point-like objects which lie either within a plane or along a straight line, the dragger's orientation will be aligned with the points, as described in Section 5.2. The default value is `false`.

**navigationPanelLines**

If `true`, causes lines to be drawn linking components in the navigation panel (Section 4.2).

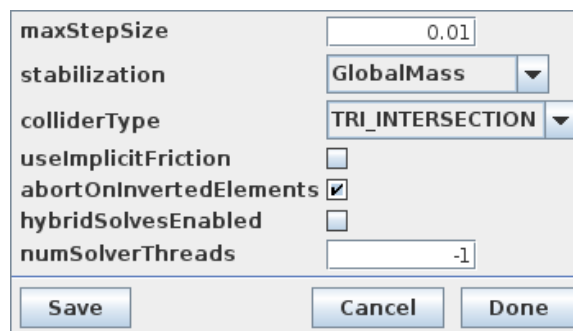
**11.1.2 Simulation**

Figure 50: Simulation settings dialog.

Simulation settings involve attributes related to model simulation. They can be adjusted by selecting “Simulation ...” from the Settings menu, which opens a *simulation settings dialog* (Figure 50). Clicking the Save button will save the current settings to the user’s preferences (Section 11.2) so that they will be set automatically when ArtiSynth is restarted.

Note that simulation settings do not take effect until the next model is loaded or reloaded.

*Simulation* settings include:

**maxStepSize**

The default maximum step size used for simulation, in seconds. See “Simulation control properties” in the [ArtiSynth Modeling Guide](#).

**stabilization**

The default stabilization method used to correct collision interpenetrations and drift from position constraints. See “Simulation control properties” in the [ArtiSynth Modeling Guide](#).

**colliderType**

The default collider type used for collision detection. See “Collision methods and collider types” in the [ArtiSynth Modeling Guide](#).

**useImplicitFriction**

The default setting for whether implicit friction should be used. See “Implicit Friction” in the “Contact and Collision” chapter of the [ArtiSynth Modeling Guide](#). Use of implicit friction can also be enabled from the command line with the option

```
-useImplicitFriction
```

### abortOnInvertedElements

This is an attribute of 3D FEM models that instructs the simulation to abort whenever inverted elements are encountered by FEM materials that don't support element inversion (such as most hyperelastic materials). The default value for this attribute is `true`. Setting it to false may be useful for diagnostic purposes.

### hybridSolvesEnabled

Enables *hybrid solves* in the sparse matrix solver used for system simulation. The default value is `true`. Hybrid solves combine iterative and direct methods in a way that can often improve simulation speeds several fold. However, hybrid solves do not generally produce results with exact numerical repeatability. It may therefore be preferable in some diagnostic and testing situations to disable hybrid solves.

### numSolverThreads

Number of threads used by the sparse matrix solver used for system simulation. The default value is the number of threads available on the machine. Simulations will usually run faster with more threads (although the speed increase is generally sublinear). However, using more than one thread will usually produce results that do not have exact numerical repeatability. It may therefore be preferable in some diagnostic and testing situations to set the number of solver threads to 1.

## 11.2 Preferences

*Preferences* are settings saved in the user's configuration data so that they are restored when ArtiSynth is restarted. They are stored in the file `settings/preferences` within the configuration folder (Section 1.1).

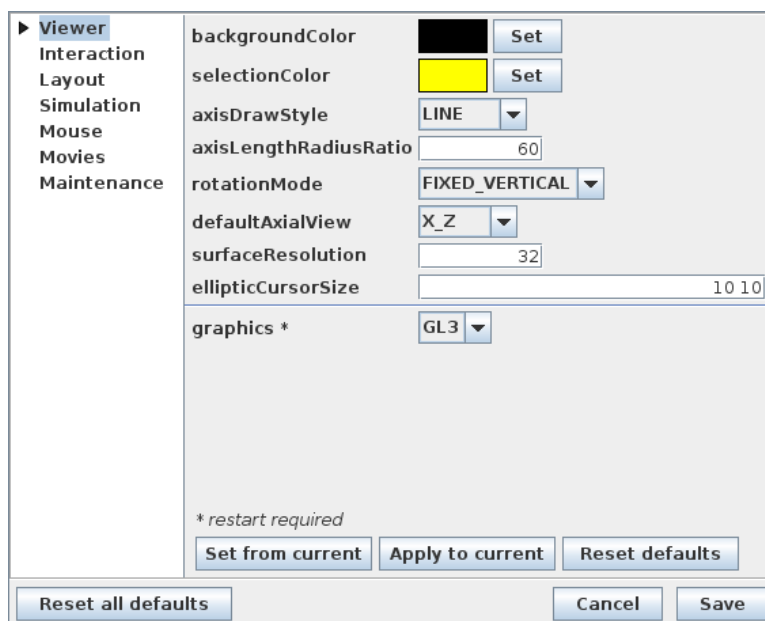


Figure 51: Preferences editor, with viewer preferences shown.

Preferences can be edited by selecting “Preferences ...” from the Settings menu, which opens a *preferences editor* dialog (Figure 51). A panel on the left contains a tree-based representation of the different preferences, which can be used to select a specific set of preferences which can then be adjusted using a *preferences panel* displayed on the right.

Values shown in the preferences editor describe stored values that are set when ArtiSynth is restarted, and setting them will *not* change the current settings in the ArtiSynth application. However, most preferences panels contain the button “Apply to current” which *will* apply the values to the current application settings. Other buttons include “Set from current”, which loads the values from the current application settings, and “Reset defaults”, which resets the panel values to their defaults.

To save preferences in the user's configuration data, one should click the Save button at the bottom right of the preferences editor. The “Reset all defaults” button located at the bottom left resets all preference values to their defaults; the Save button must subsequently be used to save these reset values.

Preferences that can be set include settings for the viewer and mouse (Sections 3.9 and 3.10), model interaction and simulation (Section 11.1), and making movies (Section 14.5). Preferences can also be set for the application layout, as described in the next section.

### 11.3 Layout preferences

Layout preferences control the size and visibility of different application windows, as well as the look and feel of the user interface.

#### **viewerWidth**

Width of the main ArtiSynth viewer (in pixels). Note that this and the `viewerHeight` property describe the size of the viewer itself, and not the overall application window containing the viewer. The viewer width can also be set from the command line with the option

```
-width <pixels>
```

#### **viewerHeight**

Height of the main ArtiSynth viewer window (in pixels). This can also be set from the command line with the option

```
-height <pixels>
```

#### **screenLocX**

Screen x location of the main ArtiSynth frame (in pixels). This can also be set from the command line with the option

```
-screenx <pixels>
```

#### **screenLocY**

Screen y location of the main ArtiSynth frame (in pixels, with 0 at the top of the screen). This can also be set from the command line with the option

```
-screeny <pixels>
```

#### **timelineVisible**

Whether or not the timeline (Section 7) is visible when ArtiSynth starts up. The default value is `true`. Initial timeline visibility can also be controlled with the command line options

```
-timelineVisible  
-timelineHidden
```

#### **timelineWidth**

Initial width of the timeline (in pixels). This can also be set from the command line with the option

```
-timelineWidth <pixels>
```

#### **timelineHeight**

Initial height of the timeline (in pixels). This can also be set from the command line with the option

```
-timelineHeight <pixels>
```

#### **timelineRange**

Default timeline range that is used to initialize the visible part of the timeline when a model is loaded. A value  $\leq 0$  implies that the range will be set automatically. If set to a value  $> 0$ , the actual amount of time visible may be slightly larger because of the way the timeline is scaled. The timeline range can also be set from the command line with the option

```
-timelineRange <value>
```

### timelineLocation

Initial location of the timeline with respect to the main ArtiSynth application window. This can also be set from the command line option with the option

```
-timelineLocation <loc>
```

where `loc` is one of CENTER, LEFT, RIGHT, ABOVE or BELOW.

### jythonFrameVisible

Whether or not a Jython console frame (Section 12) should be displayed when ArtiSynth starts up. The default value is `false`. A Jython console can also be requested at startup with the command line option

```
-showJythonConsole
```

### jythonLocation

Initial location of the Jython frame with respect to the main ArtiSynth application window. This can also be set from the command line with the option

```
-jythonLocation <loc>
```

where `loc` is one of CENTER, LEFT, RIGHT, ABOVE or BELOW.

### lookAndFeel

Controls the look and feel of the user interface (UI). The ArtiSynth UI is built using Java Swing, which supports pluggable look and feel. At the time of this writing, the look and feel options are:

#### DEFAULT

The default provided by the Java environment. On Windows and Linux systems, this is METAL, while MacOS provides its own proprietary look and feel.

#### METAL

The standard Swing cross-platform look and feel. On MacOS, it is necessary to specify this explicitly to avoid its native look and feel.

#### SYSTEM

The native system look and feel.

A restart is required when changing the look and feel. It is also possible to specify the look and feel from the command line using the option

```
-lookAndFeel <laf>
```

where `laf` is one of the options described above.

The SYSTEM look and feel may cause problems on some systems. For example, at the time of this writing, the Windows UI is not fully compatible with certain dialogs used to save files.

## 12 Jython Interaction and Scripting

ArtiSynth supports a Jython console that provides a command line interface to all the internal structures associated with ArtiSynth and its models. Jython ([www.jython.org](http://www.jython.org)) is a Java implementation of Python that combines Python commands and syntax with the ability to access and call all publicly accessible attributes and methods of Java objects. It can be used to interactively load, query and run models, or to run simulation scripts, either interactively (Section 12.3) or in batch mode (Section 12.6).

Use of the Jython interface currently requires that ArtiSynth is run under Java 8, which is why we recommend using this Java version. If you experience trouble running the Jython console, verify that you do in fact have Java 8 installed, and, if you are using an integrated development environment (IDE), that the IDE is also using Java 8. Details on installing Java 8 can be found in the installation guides for [Windows](#), [MacOS](#), and [Linux](#).

The syntax, language semantics, and common packages for Jython are the same as for Python 2.7, so Python 2.7 language references can be used to learn how to program in Jython.

The Jython console can be started by either

1. Choosing View > Show Jython Console in the GUI, or
2. Specifying the option `-showJythonConsole` on the command line.

The Jython console currently appears in a separate Window frame (Figure 52).

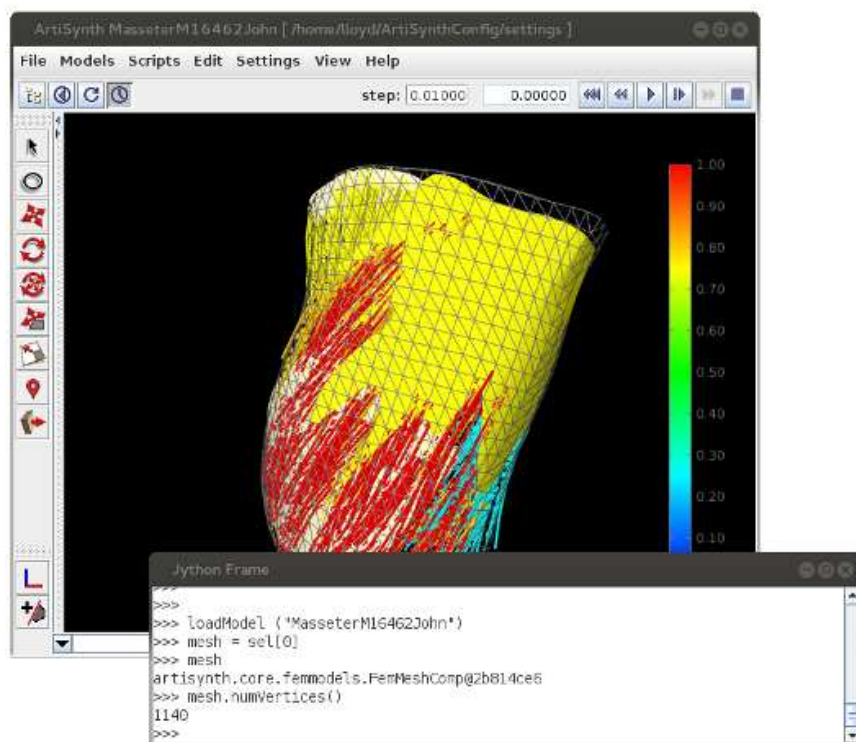


Figure 52: ArtiSynth application showing the Jython console.

## 12.1 Querying ArtiSynth structures and models

Once the Jython console is open, it can be used to query ArtiSynth structures and model components. Every publicly accessible method of every Java object can be called via the interface. To easily access particular components of a model, the predefined list variable `sel` provides access to the current ArtiSynth selection list. That means you can select items in the ArtiSynth GUI (using either the viewer or navigation panel) and then access these in Jython. By itself, `sel` supplies the entire selection list. Specific selected components can be accessed by indexing into this list, so that `sel[i]` returns the *i*-th entry (where the index *i* is 0-based).

For example, if two particles are currently selected in ArtiSynth, then `sel` can be used as follows:

```
>>> sel # get the entire selection list
[artisynth.core.mechmodels.Particle@709da188, artisynth.core.mechmodels. ←
  Particle@750eba3f]
>>>
>>> sel[0] # get the first item on the list
artisynth.core.mechmodels.Particle@709da188
```

Once a component has been selected, then one has access to all its public methods. This can be quite useful for setting or querying items that are not normally available via the ArtiSynth GUI. For example, if we want to find the number of nodes in a `FemModel3d`, then we can select the FEM and then in the console do

```
>>> fem = sel[0]
>>>
>>> fem.numNodes()
16
```

## 12.2 Object creation and importing classes

Java objects can also be created by calling their constructors directly, without the need for the keyword `new`. For example,

```
>> vec = Vector3d (1, 2, 3)
```

creates a new instance of `Vector3d` and initializes it to (1, 2, 3).

As with Java code, Java class definitions need to be imported into Jython in order for them to be visible. For example,

```
>> import maspack.matrix
```

will import all classes defined in the package `maspack.matrix`. However, unlike in Java, classes imported into Jython this way will still need to be accessed using their fully qualified class names (e.g., `maspack.matrix.Vector3d`, `maspack.matrix.Matrix3d`, etc.). In order to make classes visible using only their basic names, one may use the `from` statement, as in

```
>> from java.io import File
>> from maspack.matrix import Vector3d
```

To import *all* classes within a package by basic name, one may employ the wildcard `*`:

```
>> from maspack.matrix import *
```

although this may occasionally miss certain classes.

For convenience, the ArtiSynth Jython console already fully imports, by basic name, the classes from the following packages:

```
maspack.util
maspack.matrix
maspack.geometry
maspack.collision
maspack.render
maspack.solvers
artisynth.core.mechmodels
artisynth.core.femmodels
artisynth.core.materials
artisynth.core.modelbase
artisynth.core.driver
artisynth.core.workspace
artisynth.core.inverse
java.lang
java.io
```

## 12.3 Running simulations and scripting

Jython can also be used to run simulations, using various built-in functions that allow models to be loaded and run. A full summary of these is given in Section 12.7. In particular, it is possible to load a model and then run or single step a simulation.

To load a model, one may use the function

---

```
loadModel (name, args...)
```

where `name` is the classname of the model's [RootModel](#), and `args...` is an optional variable-length list of string arguments that are used to form the `args` argument of the model's `build()` method (see the section “Implementing the `build()` method” in the [ArtiSynth Modeling Guide](#)). In general, the `name` argument should be the fully qualified name of the root model class, as in

```
>> ah.loadModel ('artisynth.demos.mech.RigidBodyDemo')
```

However, if the model has been previously loaded by ArtiSynth, the class's simple name should work as well:

```
>> ah.loadModel ('RigidBodyDemo')
```

Once loaded, simulation may be controlled using functions such as `play()`, `pause()`, `step()`, or `reset()`. The `play()` function may take a time argument indicating the length of time the simulation should be run for; if this argument is omitted, the simulation will run indefinitely or until a breakpoint is encountered. Play requests are issued asynchronously; to make the Jython console wait for simulation to halt, one may use the `waitForStop()` function:

```
>> loadModel ('RigidBodyDemo')
>> play (2.5)
>> waitForStop()
```

When controlling simulations, it is often easiest to create a script of Jython commands in a Python-style `.py` file and then “source” them into the Jython console. In Python, one can use `exec()` or `execfile()` to do this. However, in ArtiSynth it is recommended to use the ArtiSynth supplied `script()` function, as in

```
>>> script ('contactTest.py')
```

This is particularly true for longer scripts, since `script()` interacts better with the GUI and allows the script commands to be displayed in the console as they are being executed.

Scripts are particularly useful for running multiple simulations with varying inputs and outputs. Arguments supplied to a model's `build()` method provide a convenient way to adjust input and output parameters between simulations.

As will be seen below, it is also possible to pass command-line style arguments to the script itself. Within the script, such arguments can be retrieved from `sys.argv`, as illustrated by the following code fragment:

```
import sys
print('Number of arguments:' + str(len(sys.argv)))
print('Argument List:' + str(sys.argv))
```

Jython commands, including scripts, can be aborted by clicking the stop-all button to the right of the play controls (Section 7.2.1).

## 12.4 Using the script menu

For convenience, ArtiSynth provides a Scripts menu in the main application menu bar that is similar in function to the Models menu and can be used to run script files from predefined menu entries (Figure 53). By default, the upper part of this menu contains a single submenu:

**Demo scripts** - expands to all scripts within `src/artisynth/demos/scripts` under the ArtiSynth install folder.

Selecting a submenu entry will cause the Jython console to be opened (if necessary) and the associated script to be executed. At the time of this writing, “Demo scripts” contains a single entry for a demonstration script named `demoScript.py`.

The lower part of the script menu, beneath the separator, contains entries for reloading recent scripts (“Run recent ...”), running a script from an explicitly specified file (“Run script ...”, Section 12.5), and adding custom entries to the upper part of the script menu (“Edit menu ...”, Section 13).

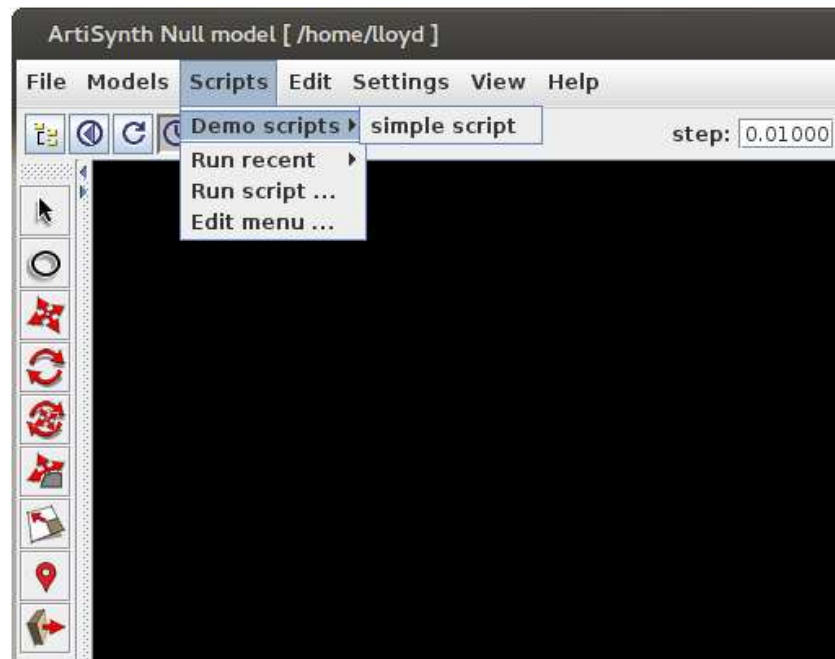


Figure 53: The ArtiSynth script selection menu.

## 12.5 Selecting a script file

To explicitly specify a script file, choose “Run script ...” from the lower part of the script menu, which will bring up a script selection dialog as shown in Figure 54.

Users should specify the folder containing the script in the “Script folder” field at the top. This folder will then be searched for files ending in `.py` and `.jy`, with the results listed in the “Script file” panel, from which the user can then select the desired script by clicking on it. If the script requires command-line style arguments (Section 12.3), these can be entered in the “Args” field near the dialog bottom. Arguments should be separated by white space, with those containing white space placed between with double quotes `“”`.

When all desired settings have been made, the user runs the script using the Run button.

## 12.6 Specifying scripts on the command line

It is possible to specify Jython scripts directly on the ArtiSynth command line using the `-script` option. For example,

```
artisynth -script experiment.py
```

will start ArtiSynth and then immediately invoke the script `experiment.py`. Scripts can also be run in “batch” mode, *without* starting the GUI or explicitly opening the Jython console. This can be useful when running ArtiSynth remotely, or in parallel on a cluster of machines. To run a script in batch mode, simply add the `-noGui` command line option:

```
artisynth -noGui -script experiment.py
```

Since there is no GUI, Jython will then be initiated using a terminal console instead of the usual GUI-based text window. When the script finishes, the console will remain available for interactive operation.

One may also simply start with a Jython console, with no initial script:

```
artisynth -noGui -showJythonConsole
```

Finally, arguments may be passed to scripts invoked using `-script`, by placing them immediately after the script specification, enclosed within square brackets `[ ]`. For example,

```
artisynth -script myscript.py [ -xxx 123 -off ]
```

will pass the strings `“-xxx”`, `“123”` and `“-off”` to the script `myscript.py`. Within the script, these arguments can be retrieved from `sys.argv`, as described in Section 12.3.

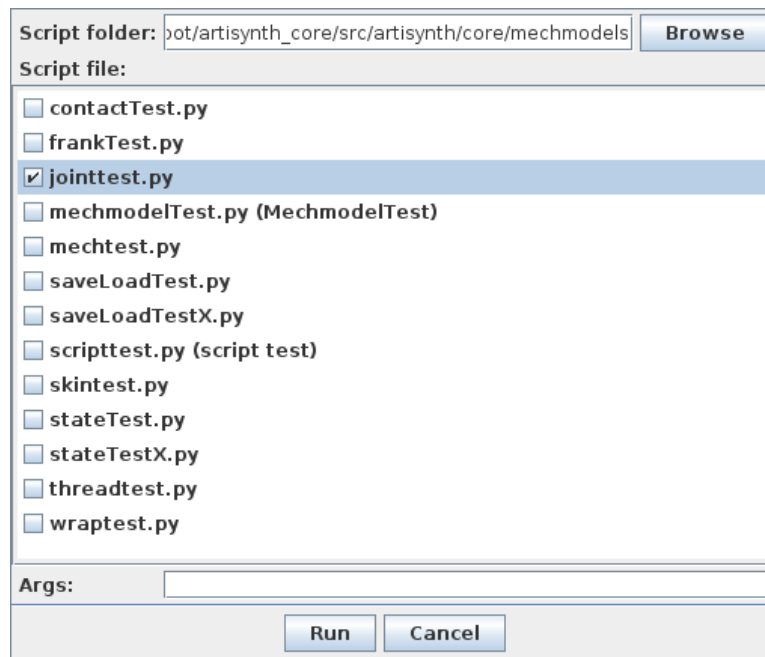


Figure 54: Dialog for selecting a script file.

## 12.7 Built-in functions

The following is a summary of the built-in Jython console functions:

`getMain()`

Returns the ArtiSynth Main object.

`loadModel (name, args...)`

Loads the named model along with optional arguments. `name` is the classname of the model's [RootModel](#) and `args...` is an optional variable-length list of string arguments that are used to form the `args` argument of the model's `build()` method. Nominally, `name` should be a fully-qualified classname (for example, `artisynth.demos.mech.RigidBodyDemo`), but if the model has been loaded before, the simple class name should work as well (e.g., `RigidBodyDemo`).

`loadModelFile (filename)`

Loads a model from an ArtiSynth file. `filename` is a string giving the file's path.

`saveModelFile (filename)`

Save a model to an ArtiSynth file.

`saveModelFile (filename, saveWayPointData, coreCompsOnly)`

Save a model to an ArtiSynth file. The options `saveWayPointData` and `coreCompsOnly` specify whether to (a) save waypoint data, and (b) save only components from `artisynth_core`.

`play()`

Starts the simulation running.

`play(t)`

Starts and runs the simulation for `t` seconds.

`pause()`

Pauses the simulation.

`step()`

Single steps the simulation.

`reset()`

Resets the simulation to time 0.

`forward()`

Forwards the simulation to the next waypoint.

`rewind()`

Rewinds the simulation to the last waypoint.

`delay(t)`

Delays execution for `t` seconds.

`waitForStop()`

Blocks until the simulation has completed.

`isPlaying()`

Returns `true` if simulation is still running.

`getTime()`

Returns current ArtiSynth simulation time in seconds.

`reload()`

Reloads the current model.

`addWayPoint(t)`

Adds a simulation waypoint at time `t`, where `t` is a floating point value giving the time in seconds.

`addBreakPoint(t)`

Adds a breakpoint at time `t`.

`removeWayPoint(t)`

Removes any waypoint or breakpoint at time `t`.

`clearWayPoints()`

Removes all waypoints and breakpoints.

`saveWayPoints (filename)`

Save waypoints and their data to a specified file

`loadWayPoints (fileName)`

Load waypoints and their data from a specified file

`root()`

Returns the current `RootModel`.

`find (path)`

Finds a component defined by `path` with respect to the current `RootModel`.

`getsel()`

Returns the current ArtiSynth selection list (which is the same as the built-in variable `sel`).

`getsel(i)`

Returns the `i`-th selection list item (where `i` is 0-based).

`quit()`

Quits ArtiSynth.

---

## 13 Customizing the Model and Script Menus

The model and script menus are useful interfaces for loading ArtiSynth models and running Jython scripts. The upper sections of these contain menu arrangements which allow models to be loaded or scripts to be run by simply clicking on a menu item. These menu arrangements may be customized by the user.

The upper model menu comes initialized with predefined entries that locate all models defined in the packages `artisynt.demos` and `artisynt.models`. Customization may be desirable in order to:

- Access models declared in packages outside `artisynt.demos` and `artisynt.models`;
- Reduce the scope of the menu to a smaller set of models or model packages.

Meanwhile, the upper script menu comes initialized to find only scripts located in `src/artisynt/demos/scripts` relative to the ArtiSynth installation folder, so customization will be necessary to access scripts outside of this.

Either menu may be customized by selecting “Edit menu ...” at the menu’s bottom, which will open one of the menu editors described below.

### 13.1 Model menu editor

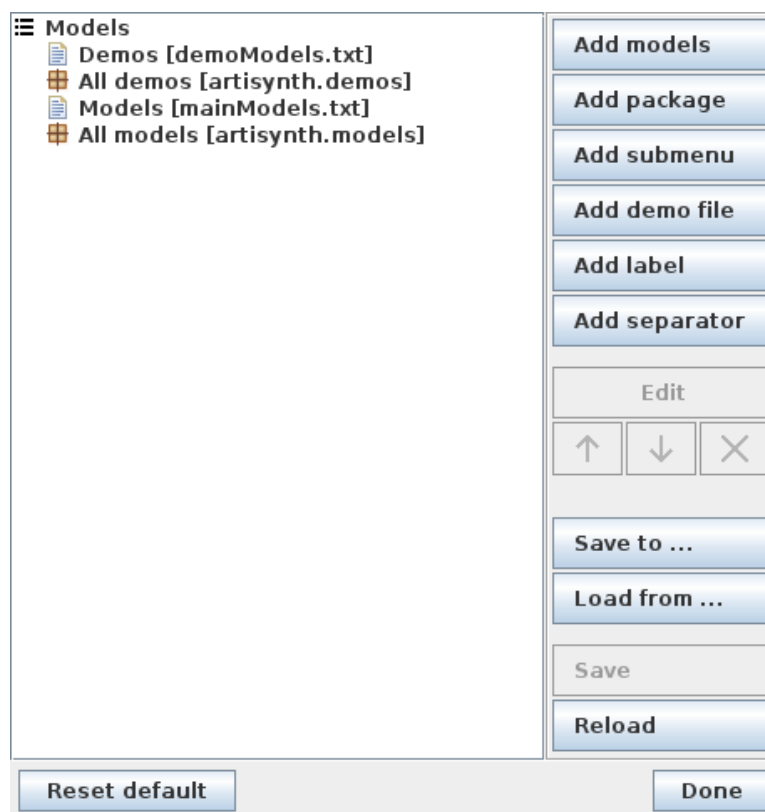


Figure 55: Model menu editor, showing the default menu configuration.

The upper part of the model menu may be customized using the *model menu editor* (Figure 55), which may be opened by selecting “Edit menu ...” at the bottom of the model menu. The editor comprises a large *menu panel* on the left side, containing a tree representation of the menu, and an *edit panel* on the right side which supports various editing actions.

A model menu is composed of a hierarchical arrangement of six types of menu entries, summarized in Table 1 and described in more detail in Section 13.3. Every entry has a *title* attribute denoting the text used in the actual menu. The set of menu entries is displayed hierarchically in the editor’s menu panel, with each entry indicated by an icon (see Table 1) followed by its title. Entries can be selected by clicking on them using the left mouse button, using the CTRL and

Icon	Type	Description
	Package	Submenu containing all models within a specific Java package and subpackages.
	Model	Menu entry for an individual model.
	Submenu	General submenu containing other menu entries.
	Demo file	Submenu consisting of all models listed in a text file.
	Label	Menu entry consisting of a descriptive label.
	Separator	Menu entry consisting of a separating line.

Table 1: The six different types of menu entries used to form a model menu.

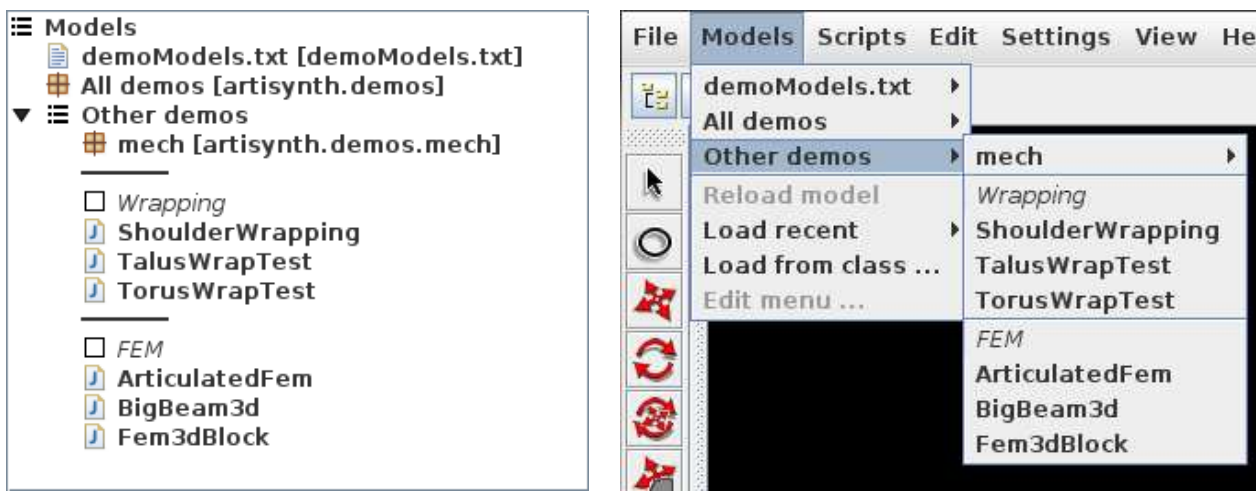


Figure 56: Model menu editing example, showing the display in the menu panel (left) and the corresponding model menu (right).

SHIFT modifier keys to enable multiple and contiguous selection. Figure 56 shows an example menu panel display on the left with the corresponding model menu on the right.

Editing actions that may be initiated include:

### Adding menu entries

The Add buttons in the edit panel allow the different menu entries to be added to the menu. Add operations for all entry types except separators invoke a dialog for configuring the new entry; details on these are given in Section 13.3. If no entries are selected when the add operation is initiated, the new entry will be placed at the end of the top-level menu. Otherwise, if a submenu entry is selected, the new entry will be placed at the end of that submenu. Finally, if a non-submenu entry is selected, the new entry will be inserted in the location of the selected entry.

### Editing menu entries

All menu entries except separators have attributes that can be edited by selecting the entry and clicking the Edit button in the editing panel. Details on these attributes are given in Section 13.3.

### Rearranging and deleting menu entries

Menu entries can be rearranged or deleted using the up/down arrow and X buttons:



Rearrangement can be done within a submenu by selecting a contiguous set of entries within that submenu and using the up/down arrows to move them up or down with respect to other submenu entries. Entries can be deleted by selecting any number of them, within any submenu, and clicking the X button.

### Saving and loading from files

The Save and Reload buttons at the bottom of the edit panel can be used to save/reload the current menu to/from the initialization file `settings/modelMenu.xml`, located in the user configuration folder (Section 1.1). The buttons "Save to ..." and "Load from ..." can be used to save/load the menu from alternative files. The "Reset to default" button in the lower left corner of the dialog resets the menu and its configuration files to their original settings.

Changes that are made in the menu editor appear immediately in the model menu itself. Changes which involve adding, deleting or rearranging entries can be undone using the Undo option in ArtiSynth's main Edit menu (Section 16.1.3).

Submenus which are empty are displayed in the menu editor but *not* in the menu itself. Submenus may be empty if no entries have been added to them, or if the model entries specified within them are not actually found. The latter most commonly occurs when the packages containing said models are not visible to ArtiSynth.

## 13.2 Script menu editor

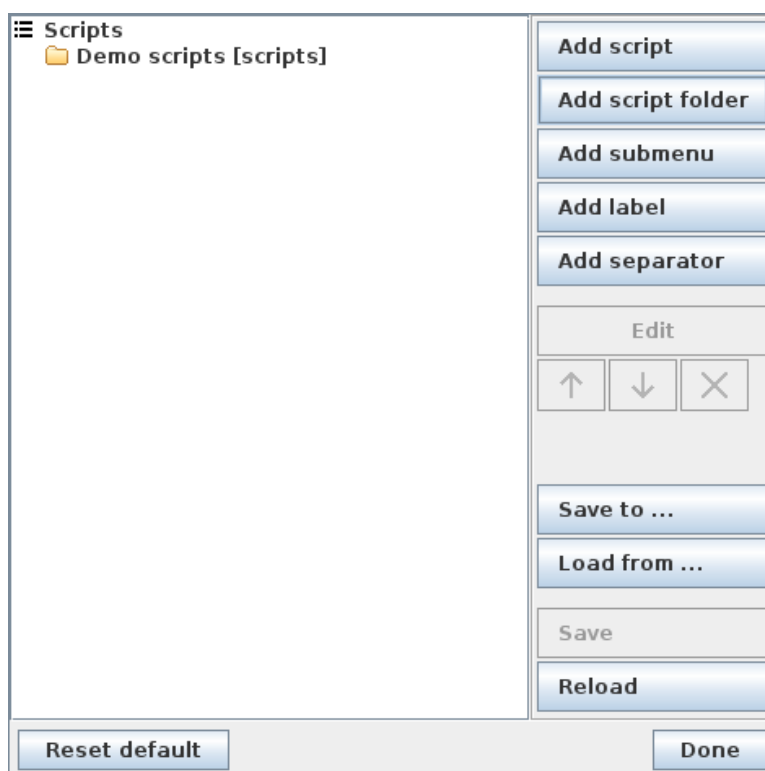


Figure 57: Script menu editor, showing the default menu configuration.

The upper part of the script menu may be customized using the *script menu editor* (Figure 57), which may be opened by selecting "Edit menu ..." at the bottom of the script menu. Its structure and function is essentially identical to the model menu editor (Section 13.1), except that the Package, Model, and Demo file entries are replaced by the Script folder and Script entries, as shown in in Table 2 and described in more detail in Section 13.3. The initialization file accessed by the Save and Reload buttons is `settings/scriptMenu.xml`, located in the user configuration folder (Section 1.1).






Icon	Type	Description
	Script folder	Submenu containing all scripts within a specific folder.
	Script	Menu entry for an individual script.
	Submenu	General submenu containing other menu entries.
	Label	Menu entry consisting of a descriptive label.
	Separator	Menu entry consisting of a separating line.

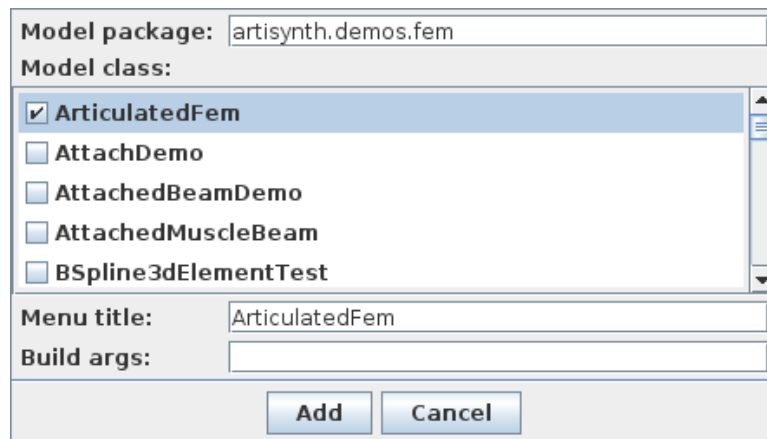
Table 2: The five different types of menu entries used to form a script menu.

## 13.3 Menu entry types

### 13.3.1 Model

For model menus, a model entry is used to select a specific model; clicking on this entry type in the model menu causes the class defining the model to be loaded. Model classes must be public subclasses of [RootModel](#).

To add a model entry to the menu, click **Add model** in the edit panel. This opens an *add model* dialog,



which allows the user to specify the model class and (optional) `build()` method arguments using the “Model package”, “Model class” and “Build args” fields in the same manner discussed in Section 2.2. Multiple models may be selected in the “Model class” panel; if this is done, a separate model entry will be created for each. The menu title can be set explicitly using the “Menu title” field; otherwise, the title will default to the model class’s simple name.

Once created, a model entry can be edited by selecting it in the editor’s menu panel and clicking the **Edit** button, which opens an editing dialog that also allows the user to adjust the title font properties.

### 13.3.2 Package

For model menus, a package entry is a submenu containing *all* the model classes found within a specified Java package and its subpackages. The submenu contents may be arranged either as a flat list, or hierarchically based on the subpackages (the default). Again, each model class must be an public subclasses of [RootModel](#). Individual model classes can be *excluded* from package menu entries by defining the public `boolean` member variable `omitFromMenu` and setting it to `true`:

```
public class MyModel extends RootModel {
    ...
    public static boolean omitFromMenu = true;
    ...
}
```

To add a package entry to the menu, click “**Add package**” in the edit panel. This opens an *add package* dialog,



Package:	artisynt.demos.mech
Menu title:	DemosMech
Flat view:	<input type="checkbox"/>
<div>Add Cancel</div>	

which allows the user to specify the package, menu title, and whether to use a flat or hierarchical view. Auto-completion is supported by the Package field, with the <TAB> character invoking auto-completion based on currently known packages and repeated use of either <TAB> or the up/down arrows will scroll through known packages. Package entry is completed using the <ENTER> key.

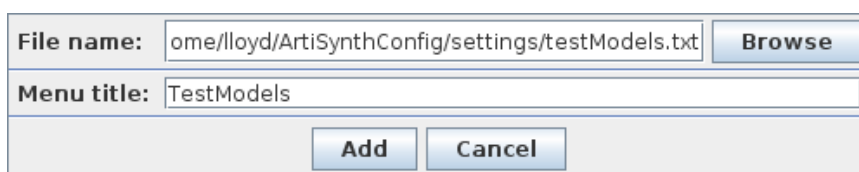
If the title is omitted (via an empty “Menu title” field), then the menu entries are not presented as a submenu, but instead are unrolled directly into the submenu containing the package entry.

Once created, a package entry can be edited by selecting it in the editor’s menu panel and clicking the Edit button. This opens an editing dialog that allows the user to adjust additional attributes, such as the title font properties, the maximum number of rows, and whether to employ scrolling, as described in more detail in Section 13.3.6.

### 13.3.3 Demo file

For model menus, a “demo file” entry is a submenu containing all the models listed in a plain text file, using the format described in Section 13.5. Plain text files have the advantage of being more human-readable, and are easier to edit and comment out line.

To add a demo file entry to the menu, click “Add demo file” in the edit panel. This opens an *add demo file* dialog,



File name:	ome/lloyd/ArtiSynthConfig/settings/testModels.txt	Browse
Menu title:	TestModels	
<div>Add Cancel</div>		

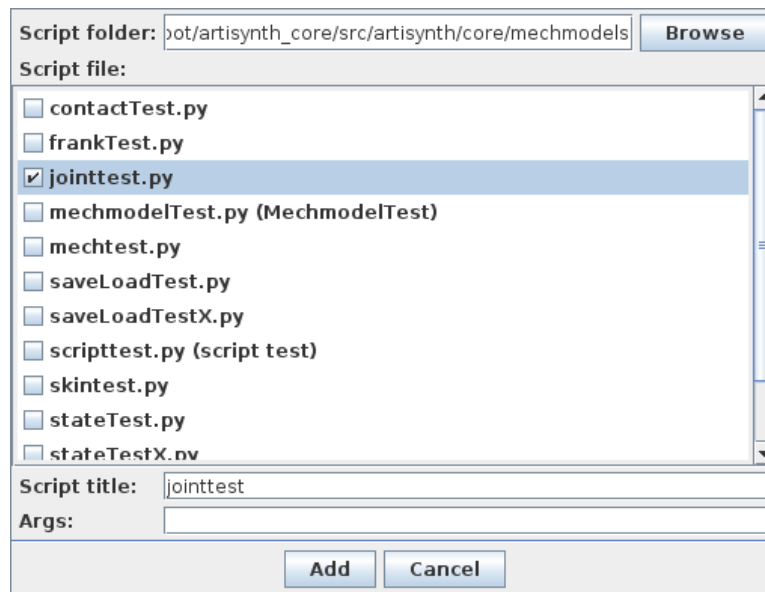
which allows the user to specify the file name and the menu title. If the title is omitted (via an empty “Menu title” field), then the menu entries are not presented as a submenu, but instead are unrolled directly into the submenu containing the entry.

Once created, a demo file entry can be edited by selecting it in the editor’s menu panel and clicking the Edit button. This opens an editing dialog that allows the user to adjust additional attributes, such as the title font properties, the maximum number of rows, and whether to employ scrolling, as described in more detail in Section 13.3.6.

### 13.3.4 Script

For script menus, a script entry is used to select a specific script file; clicking on this entry type in the script menu causes this script file to be run.

To add one or more script entries to the menu, click “Add scripts” in the edit panel. This opens an *add script* dialog,



which allows the user to specify the script and (optional) arguments using the “Script folder”, “Script file”, and “Args” fields in the same manner discussed in Section 12.5. Multiple scripts may be selected in the “Script file” panel; if this is done, a separate entry will be created for each. The menu title can be set explicitly using the “Menu title” field; otherwise, the title will default to the script’s file name.

Once created, a script entry can be edited by selecting it in the editor’s menu panel and clicking the Edit button, which opens an editing dialog that also allows the user to adjust the title font properties.

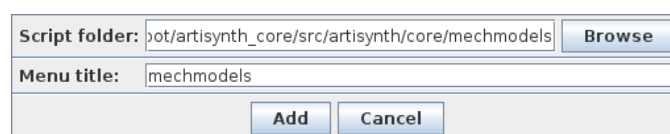
### 13.3.5 Script folder

For script menus, a script folder entry is a submenu containing *all* the scripts found within a specified folder. The scripts are found by searching the folder for all files that end in `.py` and `.jy`, and which also contain the special first line

```
# ArtisynthScript : "scriptTitle"
```

where `scriptTitle` is the desired menu title for the script.

To add a script folder entry to the menu, click “Add script folder” in the edit panel. This opens an *add script folder* dialog,



which allows the user to specify the script folder and menu title via the “Script folder” and “Menu title” fields. If not explicitly specified, the title defaults to the name of the folder. If the title is omitted, by explicitly setting it to an empty string, then the menu entries are not presented as a submenu, but instead are unrolled directly into the submenu containing the script folder entry.

Once created, a script folder entry can be edited by selecting it in the editor’s menu panel and clicking the Edit button. This opens an editing dialog that allows the user to adjust additional attributes, such as the title font properties, the maximum number of rows, and whether to employ scrolling, as described in more detail in Section 13.3.6.

### 13.3.6 Submenu

A submenu entry is a generalized submenu that can contain an arbitrary list of other menu entries, and is used to create hierarchical menus.

To add a submenu entry to the menu, click “Add submenu” in the edit panel. This opens an *add submenu* dialog,

which allows the user to specify the submenu's title. Once the submenu is created, new menu entries can be added to it by selecting the submenu in the editor's menu panel and then clicking the appropriate Add buttons; the new entries will be placed at the end of the submenu. Alternatively, if the user selects a non-submenu entry within the submenu, and then clicks one of the Add buttons, the new entry will be placed in the location indicated by the selection.

Once created, a submenu entry can be edited by selecting it and clicking the Edit button, which opens an editing dialog that allows the user to adjust additional attributes, including whether to support scrolling, the maximum numbers of rows in the menu, and the title font properties:

Scrolling can be useful for submenus with many entries: when the number of rows exceeds the specified maximum, then the menu is presented within a scrolling pane. Otherwise, if scrolling is not enabled, extra entries are accommodated by adding additional columns to the menu (Figure 58).



Figure 58: When the number of menu entries exceeds the maximum number of rows, either a scrolling menu is used if scrolling is enabled (left), or additional columns are added to the menu (right).

Both the package, demo file and script folder entries (Sections 13.3.2, 13.3.3 and 13.3.5) are also themselves submenus, and so the scrolling and maximum rows attributes can be set for these as well.

### 13.3.7 Label

A label entry is an inactive text element that cannot be selected in the menu. It can be used to label a group of entries. To help distinguish a label from an active menu item, the default font style is set to “italic”.

To add a label entry to the menu, click “Add label” in the edit panel. This opens an *add label* dialog,

in which the user enters the label title. After it has been created, the label can be edited by selecting it in the editor's menu panel and clicking Edit, which opens an editing dialog that allows the user to also adjust the title font properties.

### 13.3.8 Separator

A separator entry is a horizontal line that visually separates menu entries. It can help distinguish groups of related menu items.

To add a separator entry to the menu, click "Add separator" in the edit panel. This will simply add a separator at the location indicated by the current selection; no dialog is needed.

## 13.4 Command line options

It is also possible to explicitly set the model menu using ArtiSynth command line arguments:

### **-modelMenu <xmlFileName>**

Specifies the model menu using an XML file, the format for which is described in Section 13.6. When this is done, the Save and Reload operations described in Section 13.1 are redirected to the specified file instead of the default configuration file.

### **-demoFile <fileName>**

Specifies a basic menu from a simple list of models provided by <fileName>, using the format described in Section 13.5. When this option is used, the model menu editor is not available and the resulting menu cannot be edited.

## 13.5 Demo file text format

A plain text file format is used to supply a list of model menu items, for either demo file menu entries (Section 13.3.3), or use with the `-demoFile` command line option (Section 13.4). In this format, entries are listed as title-class pairs, separated by whitespace. Lines beginning with a hash (#) are ignored. Titles containing spaces must be surrounded by quotation marks. The following is an example:

```
# Inverse Demos
HydrostatInvDemo artisynth.models.inversedemos.HydrostatInvDemo
"Tongue tracking" artisynth.models.inversedemos.TongueTip
```

## 13.6 XML Menu Format

Model and script menus are stored using a custom XML file format, which is documented here in case users wish to create their own files from scratch. The default files specifying the model and script menus are `modelMenu.xml` and `scriptMenu.xml`, located in the `settings` subfolder of the configuration folder (Section 1.1).

XML schemas are provided that describe and enforce the required document structure. The schemas for the model and script menus are located, respectively, in

```
src/artisynth/core/modelmenu/modelmenu.xsd
src/artisynth/core/modelmenu/scriptmenu.xsd
```

relative to the ArtiSynth installation folder. The following XML elements are defined:

For model menu files:	
ModelMenu:	the root element of the document
model:	specifies a specific model
package:	finds and lists all models in a specific Java package
demoFile:	specifies a submenu of models listed in a text file
For script menu files:	
ScriptMenu:	the root element of the document
script:	specifies a specific script file
scriptFolder:	finds and lists all scripts in a specific folder
For either file:	
menu:	creates a submenu
label:	inserts an inactive text entry
separator:	inserts a horizontal line that separates entries
hidden:	convenience element for hiding menu entries (i.e. commenting them out)

Detailed descriptions of these element types and their supported attributes are provided in the following sections.

Attributes of the `model`, `package`, `demoFile`, `script`, `scriptMenu`, `submenu`, `demoFile`, `label`, and `separator` types correspond to attributes of the menu entry types described in Section 13.3. Attributes marked by an asterisk are required in the element definition.

### 13.6.1 The root elements

The root elements `ModelMenu` and `ScriptMenu` encapsulate the entire description for either a model or script menu. They must make reference to the schema for validation purposes. The following code snippet can be used as a template for creating a model menu file,

```
<?xml version="1.0" encoding="UTF-8"?>
<ModelMenu xmlns="http://www.artisynth.org"
  xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
  xsi:schemaLocation=
    "http://www.artisynth.org src/artisynth/core/modelmenu/modelmenu.xsd">
  ... contents of menu ...
</ModelMenu>
```

while the next snippet can be used for script menus:

```
<?xml version="1.0" encoding="UTF-8"?>
<ScriptMenu xmlns="http://www.artisynth.org"
  xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
  xsi:schemaLocation=
    "http://www.artisynth.org src/artisynth/core/modelmenu/scriptmenu.xsd">
  ... contents of menu ...
</ScriptMenu>
```

### 13.6.2 Model element

Describes a model menu entry (Section 13.3.1), with the `class` attribute specifying the model class, which must be a public subclass of `RootModel`.

#### Attributes:

<code>class*</code> :	class to load when the entry is selected
<code>buildArgs</code> :	command-line style arguments to be passed to the model's <code>build()</code> method
<code>title</code> :	title text for menu entry (default: the class name)
<code>fontname</code> :	font family for the title text
<code>fontstyle</code> :	font style, from { "", "bold", "italic", "bold italic" }
<code>fontsize</code> :	font size

The following snippet creates a menu entry titled "Muscle Arm", which will launch the model `FemMuscleArm`.

```
<model class="artisynth.models.femdemos.FemMuscleArm" title="Muscle Arm" />
```

### 13.6.3 Package element

Describes a package entry (Section 13.3.2), with the `name` attribute specifying the Java package containing the model classes.

#### Attributes:

<code>name*</code> :	Name of the Java package in which to search for models
<code>title</code> :	title text for menu entry; if omitted, entries are unrolled into parent menu
<code>view</code> :	display format {"flat", "hierarchical"} (default: "hierarchical")
<code>scrolling</code> :	enables a scrolling menu (default: false)
<code>maxRows</code> :	maximum number of rows in the submenu (default: 20)
<code>compact</code> :	level of compactness {0, 1, 2} (default: 0)
	0: A new submenu is created for each subpackage (hierarchical), displayed text refers to full package.class name relative to source (flat)
	1: subpackages containing a single entity are merged into the parent menu (hierarchical), displayed text refers to unique part of the package.class name only (flat)
	2: subpackages containing a single entity are merged into the parent menu (hierarchical) and displayed text refers to the class name only (hierarchical/flat)
<code>fontname</code> :	font family for the title text
<code>fontstyle</code> :	font style, from {"", "bold", "italic", "bold italic"}
<code>fontsize</code> :	font size

If the `title` attribute is omitted, then the submenu contents are unrolled directly into the parent menu.

If font information is provided, it is applied to all entries. The following snippet creates a submenu containing every instance of `RootModel` found in the package `artisynth.demos.mech`, with a small font specified for all entries to get the menu to fit easily on the screen:

```
<package title="mech" name="artisynth.demos.mech" fontsize="5"/>
```

### 13.6.4 DemoFile element

Describes a demo file entry (Section 13.3.3), with the `file` attribute specifying the path to the file listing the models. If this path is relative and not absolute, the parser tries to locate the file with respect to (in order) (a) the folder containing the XML file being read, (b) the user's home folder and (c) the ArtiSynth installation folder.

#### Attributes:

<code>file*</code> :	file listing the model entries
<code>title</code> :	title text for menu entry; if omitted, entries are unrolled into parent menu
<code>scrolling</code> :	enables a scrolling menu (default: false)
<code>maxRows</code> :	maximum number of rows in the submenu (default: 20)
<code>fontname</code> :	font family for the title text
<code>fontstyle</code> :	font style, from {"", "bold", "italic", "bold italic"}
<code>fontsize</code> :	font size

If the `title` attribute is omitted, then the submenu contents are unrolled directly into the parent menu. If font information is provided, it is applied to all model entries that are created.

The following snippet loads all models from the original default ArtiSynth Models menu.

```
<demoFile title="Demo models" file="demoModels.txt" />
```

### 13.6.5 Script element

Describes a script menu entry (Section 13.3.4), with the `file` attribute specifying the path to the script file. If this path is relative and not absolute, the parser tries to locate the file with respect to (in order) (a) the user's home folder and (b) the ArtiSynth installation folder.

**Attributes:**

<code>file*</code> :	file path for the script
<code>args</code> :	command-line style arguments to be passed to the script
<code>title</code> :	title text for menu entry (default: the base file name)
<code>fontname</code> :	font family for the title text
<code>fontstyle</code> :	font style, from { "", "bold", "italic", "bold italic" }
<code>fontsize</code> :	font size

The following snippet creates a script entry titled "run models", which will run the script `/home/lloyd/modelrunner.py`:

```
<script file="/home/lloyd/modelrunner.py" title="run models" />
```

### 13.6.6 ScriptFolder element

Describes a submenu entry containing all scripts located automatically within a specific folder, as detailed in Section 13.3.5, with the `file` attribute specifying the path to the folder. If this path is relative and not absolute, the parser tries to locate the folder with respect to (in order) (a) the user's home folder and (b) the ArtiSynth installation folder.

**Attributes:**

<code>file*</code> :	path name of the folder containing the scripts
<code>title</code> :	title text for menu entry; if omitted, entries are unrolled into parent menu
<code>scrolling</code> :	enables a scrolling menu (default: false)
<code>maxRows</code> :	maximum number of rows in the submenu (default: 20)
<code>fontname</code> :	font family for the title text
<code>fontstyle</code> :	font style, from { "", "bold", "italic", "bold italic" }
<code>fontsize</code> :	font size

If the `title` attribute is omitted, then the submenu contents are unrolled directly into the parent menu. If font information is provided, it is applied to all entries.

The following snippet creates a script folder entry titled "Demo scripts", based on all the script files in the folder `src/artisynth/demos/scripts`:

```
<scriptFolder file="src/artisynth/demos/scripts" title="Demo scripts" />
```

### 13.6.7 Submenu element

Describes a submenu menu entry (Section 13.3.6).

**Attributes:**

<code>title*</code> :	title text for menu entry
<code>scrolling</code> :	enables a scrolling menu (default: false)
<code>maxRows</code> :	maximum number of rows in the submenu (default: 20)
<code>fontname</code> :	font family for the title text
<code>fontstyle</code> :	font style, from { "", "bold", "italic", "bold italic" }
<code>fontsize</code> :	font size

The following code snippet creates a submenu "FEM Models", which in turn lists three specific FEM models:

```
<menu title="FEM Models">
  <model class="artisynth.demos.fem.ArticulatedFem" title="ArticulatedFem"/>
  <model class="artisynth.demos.fem.AttachDemo" title="AttachDemo"/>
  <model class="artisynth.demos.fem.BigBeam3d" title="BigBeam3d"/>
</menu>
```

### 13.6.8 Label element

Describes a label entry (Section 13.3.7).

**Attributes:**

```

title*:      title text for the label
fontname:    font family for the title text
fontstyle:   font style, from {"", "bold", "italic", "bold italic"}
fontsize:    font size

```

In the following code snippet, a label is added before the tongue tracking models.

```

<label title="Tongue Tracking Models" fontstyle="italic" />
<model class="artisynth.models.tracker.JawDynamicTongue" />
<model class="artisynth.models.tracker.JawKinematicTongue" />

```

### 13.6.9 Separator element

Describes a separator entry (Section 13.3.8). The `separator` element has no attributes.

The following snippet adds a separator line between models found in the `artisynth.models.femdemos` package and those in `artisynth.models.inversedemos`.

```

<package title="femdemos" name="artisynth.models.femdemos" />
<separator/>
<package title="inversedemos" name="artisynth.models.inversedemos" />

```

### 13.6.10 Hiding elements

It is often convenient to have the ability to “comment-out” lines in any kind of coding system: the data remains in the file, but has no effect when processed. The typical method for commenting in XML is to use the `<!-- -->` tags. Unfortunately, this can be messy, and comments cannot contain other comments. For this reason, a special `hidden` element was created. Any entries within a `hidden` element are ignored by the parser.

```

<!-- temporarily hide FEM demos -->
<hidden>
  <!-- All FEM demos -->
  <package title="FEM demos" name="artisynth.models.femdemos" view="flat"/>
</hidden/>

```

## 14 Making Movies

ArtiSynth includes the ability to make movies from simulations. This process is controlled through a *movie panel* (Figure 59) that can be opened from the main application menu by navigating through View > Show movie panel.

A simple movie creation workflow, using the default settings, is the following:

1. Open the movie panel.
2. Start the movie capture by clicking the Start button at the bottom of the movie panel. By default, this will also start the simulation, whose progress can be examined in the main ArtiSynth viewer. Because of the movie capture overhead, the simulation will usually run more slowly than usual.
3. When the desired simulation time is reached, stop the movie capture and create the movie by clicking the the Stop button.

By default, this will create a JPEG movie with a frame rate of 50 Hz, packaged in a QuickTime `.mov` file with a title derived from the model name, and place it in the *movie folder*, the default location for which is the `movies` subfolder of the configuration folder (Section 1.1).

More detailed control of the movie making process, including the frame rate and method used to create the movie, can be obtained by setting options in the movie panel’s Recorder, Encoder, and Advanced tabs, as described below.

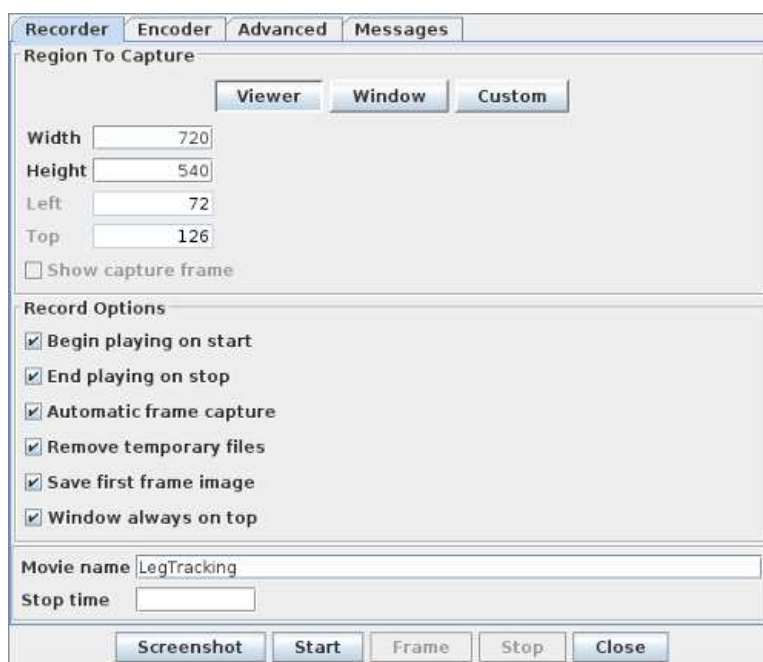


Figure 59: Movie panel, showing the Recorder tab and the default settings.

Movies produced by the default method, which creates JPEG-based movie files, tend to be rather large. If you are making movies regularly, you should consider installing an external movie making application, such as FFmpeg, and then setting the movie maker to use this via the Method field on the Encoder tab (Section 14.2). This will significantly reduce the size of the movie file and likely improve movie quality. Installing FFMEG is discussed in Section 14.6.

When capturing a movie, individual images for each frame are first saved in the movie folder and then used to create the movie itself. Each image file is named `frameXXXXX.zzz`, where XXXXX is the sequential frame number, each X is a digit, and .zzz is the image format extension (usually .png or .jpg). If “Remove temporary files” is selected in the Recorder tab of the movie panel, the image files will be removed once the movie is made. If the files are not removed, they will be overwritten by any subsequent movie creation.

Text messages describing the movie making process will be displayed in the Messages tab (Figure 60), and a popup message will indicate its completion. If an error occurs (which may happen if a non-default movie making method is selected that is not supported on the ArtiSynth host computer), information about it will be displayed in the Messages tab.

## 14.1 Recorder tab

The Recorder tab (Figure 59) controls which image region to capture along with various recording and other options. Options within the tab are organized into three groups:

### 14.1.1 Region to capture

These options define the movie capture region, as specified by three buttons: Viewer, Window, and Custom. Viewer, which is the default, sets the capture area to be the main ArtiSynth viewer window, *excluding* the surrounding menus and toolbars. This is useful for recording model demonstrations. The Window option set the capture area to be the main application window, including the main viewer and the menus and toolbars surrounding it. The Custom option allows the user to manually set the capture area to be any region on the desktop, and is useful for making movies that include the timeline, control panels, or other windows that are separate from the main application window.

When Viewer mode is selected, additional options become enabled in the Output Size section of the Encoder tabs (Section 14.2) that allow you to specify an image resolution independent of the viewer’s size. The # samples specifies

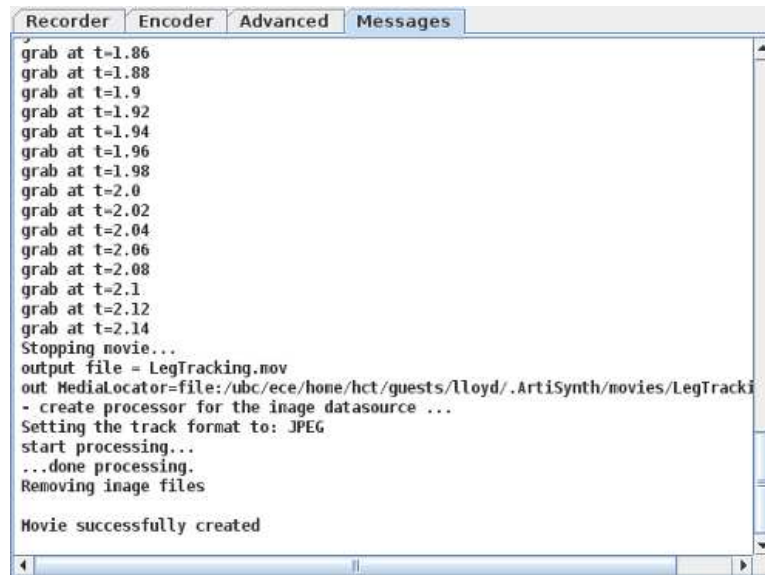


Figure 60: Messages tab of the movie panel.

the size of the multi-sample buffer, which is used for anti-aliasing. This is the only mode that continues to work correctly when a screen-saver is activated.

When Custom mode is selected, a separate *capture* window appears that can be sized and located anywhere on the desktop, with the a red outline indicating the region that will appear in the movie. The visibility of this window can be controlled by the “Show capture frame” toggle button. The capture window emulates transparency by displaying an image of the desktop beneath it; however, this image is fixed and is only refreshed when the capture window is made visible, or when one executes a double click with the left mouse button inside it.

Four fields, labeled Width, Height, and Left and Top, display the current dimensions and location of the capture region. For Viewer and Window modes, the Width and Height fields are editable and can be used to explicitly set the dimension of the viewer or main application window. For Custom mode, all four fields are editable and can be used to explicitly set the dimension and location of the capture region.

#### 14.1.2 Record options

These determine how the movie recorder works with ArtiSynth, and how it deals with the frame files.

##### Begin playing on start

When the start button is clicked, the simulation will begin to run.

##### End playing on stop

When the stop button is clicked, the simulation stops running.

##### Automatic frame capture

Frames are automatically captured according to the movie’s frame-rate while the model is run. If this is disabled, it is up to the user to click on the Frame button to capture the next frame.

##### Remove temporary files

When selected, the temporary frame images are deleted after the movie is made.

##### Save first image

This saves the first frame taken, which can be useful for representing the movie in websites.

### 14.1.3 Other options

Two other options fields are provided near the bottom of the Recorder tab:

#### Movie name

Specifies the base name of the movie output file; is set to the model name by default.

#### Stop time

When set to a positive value, indicates an explicit simulation time at which the movie should be stopped. The default value of this field is blank, indicating no specified stop time.

## 14.2 Encoder tab

The Encoder tab (Figure 61) controls encoder options and the output size.

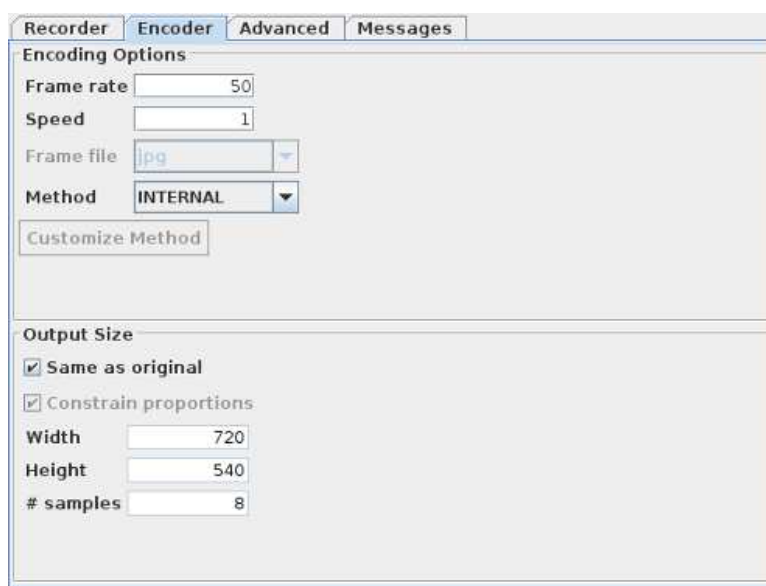


Figure 61: Encoder tab of the movie panel.

### 14.2.1 Encoder options

These describe the frame rate, relative speed, and the encoding method used to make the movie.

#### Frame rate

This is the number of frames recorded per second of movie. It is recommended to set this to be compatible with the ArtiSynth simulation step size, such that if  $r$  is the frame rate and  $h$  is the step size,  $1/r = nh$  where  $n$  is a positive integer. The default value of 50 is compatible with typical step sizes such as  $h = 0.01$  and  $0.001$ .

#### Speed

This is the ratio of the movie's speed to reality's speed. While the movie is recording, the calculations may slow down the simulation, but the movie will not be affected.

#### Frame file

This is the format the frame images will be stored in. If the internal method (see below) is used, then the frames must be stored as JPEG files.

#### Method

This describes the software used to create the movie from the captured frame images. Many of these methods require supporting software to be installed on the ArtiSynth host computer and runnable from a command line context.

**INTERNAL**

Use Java's built in support to compress the pictures into an animated JPEG.

**FFMPEG**

Use the FFmpeg command-line utility to generate the movie. Requires FFmpeg to be installed and runnable from the command line (see Section 14.6).

**MENCODER**

Use the mencoder command-line utility to generate the movie. Requires mencoder to be installed and runnable from the command line.

**ANIMATED\_GIF**

Uses an algorithm built into ArtiSynth to generate an animated gif. By hitting the Customize Method button, you can set the number of times to loop (-1 for infinity) and the frame rate (defaults to capture frame rate).

**AVCONV**

Uses the avconv utility, that is sometimes available on linux systems, to generate the movie. Requires avconv to be installed and runnable from the command line.

**CUSTOM**

Uses a custom command line utility to generate the movie. Note: the default command line specification for this option is blank, and so before using it, it must be initialized using the Customize Method button as described below.

### 14.2.2 Customizing the encoder command

The FFMPEG, MENCODER, AVCONV and CUSTOM methods described above all work by calling their respective encoders as a separate process, described using a command line specification, with the process's working directory set to the movie folder. Default command line specifications are supplied for FFMPEG, MENCODER, AVCONV, while the one for CUSTOM is blank and must be supplied by the user.

Clicking the "Customize Method" button immediately below the Method field opens a text dialog that allows the command line specification to be edited. Users may alter the command in any way desired, including modifying options or changing the name of the command itself. Several special variables beginning with \$ are used to supply information from the movie panel; before the command is executed, these are expanded to their current values:

**\$FMT**

Format of the frame image files. Expands to the value in the Frame file field of the Encoder tab.

**\$OUT**

Name of the output file. Expands to the value of the Movie name field in the Recorder tab.

**\$FPS**

Frame rate. Expands to the value in the Frame rate field of the Encoder tab.

The default movie making method, along with any method customizations, can be saved permanently in the user's preferences. See Section 14.5 for details.

## 14.3 Output size options

These options are only used when the capture area is set to Viewer and control the size of the output video, allowing the contents of the viewer to be magnified.

**Same as original**

The output video is created at the original size.

**Constrain proportions**

The output video is created with constrained proportions, such that the ratio between height and width are maintained.

**Width**

Defines the width of the output video.

**Height**

Defines the height of the output video.

**# samples**

Sets the number of samples to use for the multi-sample buffer. This only applies in Viewer mode, and is used to perform anti-aliasing.

**Note:** If your movie comes out black or only shows a section of the viewer correctly in Viewer mode, then it is likely your graphics card does not support multi-sample buffers. On machines with multiple graphics cards (e.g. laptops with both discrete and integrated graphics), make sure the Java process is set to use the discrete card. Otherwise, set # samples = 1 to disable the multi-sample buffer.

## 14.4 Advanced tab

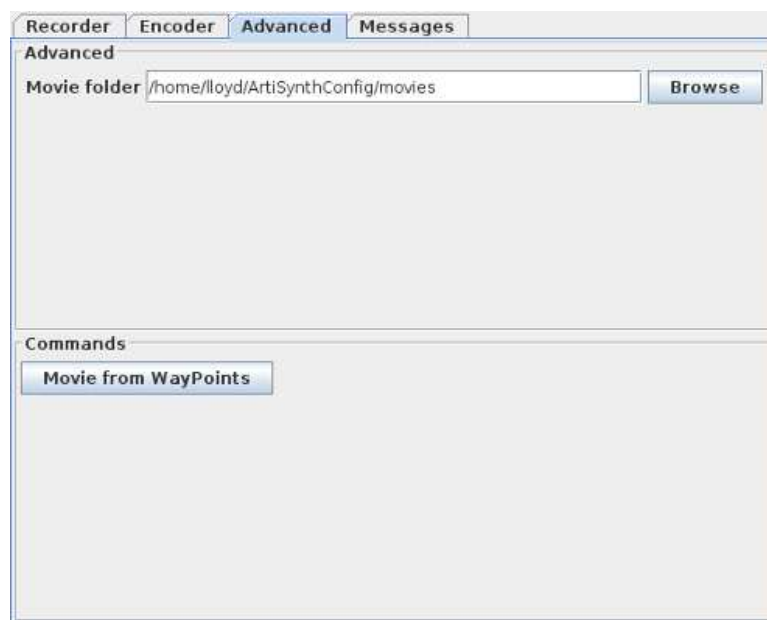


Figure 62: Advanced tab of the movie panel.

The Advanced tab (Figure 62) presently provides two options:

1. A "Movie folder" field which allows the movie folder to be customized. This folder can also be saved permanently in the user's preferences, as described in Section 14.5.
2. A "Movie from waypoints" button which creates a movie from the model's waypoint data. To work correctly, the model must have valid waypoints defined at regular intervals corresponding to the frame rate specified in the Encoder tab. The viewer image of the model at each of these waypoints is used to generate the frame image files from which the movie is then created. This enables a movie to be created from waypoint data without having to run a simulation.

## 14.5 Saving movie preferences

Some of the movie settings described above can be saved permanently in the user's preferences (Section 11.2). To do this, choose Preferences from the application Settings menu and open the Movies panel.

Preferences that can be currently saved include:

- Frame rate
- Movie method
- Movie method customization
- Movie folder

To specify a method customization, select the desired method and then click the “Customize Method” button.

## 14.6 Installing FFmpeg

As described above, movies produced by the default method, which creates JPEG-based movie files, tend to be rather large. If you are making movies regularly, you should consider installing an external movie making application such as FFmpeg, and then setting the movie maker to use this via the Method field on the Encoder tab (Section 14.2). This will significantly reduce the size of the movie file and likely improve movie quality.

At the time of this writing, FFmpeg is available for Windows, Mac and Linux through

[ffmpeg.org/download.html](http://ffmpeg.org/download.html)

The easiest option is to download a precompiled release. The FFmpeg download site does not provide these itself, but refers to other sites that do. For example, the Windows download page currently contains a link to

[Windows builds from gyan.dev](http://Windows builds from gyan.dev)

which under git master builds contains a download link for ffmpeg-git-full.7z. This can be extracted into a location such as C:\ffmpeg, after which the installation’s bin folder (e.g., C:\ffmpeg\bin) should be added to your Windows Path environment variable. How to do this is described in the section “Adding Directories to the System Path” in the Windows version of the ArtiSynth Installation Guide.

Detailed instructions for installing FFmpeg on Windows can also be found at sites such as

[phoenixnap.com/kb/ffmpeg-windows](http://phoenixnap.com/kb/ffmpeg-windows).

## 15 Control Panels

Control panels are essentially custom-built property panels that are attached to the root model and let the user interactively set or adjust various properties while the simulation is in progress. Most of the panels that appear with the various ArtiSynth demos are in fact control panels specially created for the demo in question.

The properties controlled by a control panel do not need to come from the same object; instead, they can come from a variety of objects. However, unlike with property panels, it is not possible (at the time of this writing) for a control panel widget to control a property across multiple objects.

The problem of controlling the same property in multiple objects may be addressed in future by the introduction of *component groups*.

### 15.1 Creating control panels

To create a control panel, select Edit > Add control panel from the ArtiSynth main menu. This will cause a blank control panel to appear.

To add a widget to this panel, right-click inside the the panel and select Add widget. This will cause a widget creation dialog to appear, as shown in Figure 63.

The top-most widget in this dialog is a component/property selector. The component section is a selection display identical in function to that described in Section 4.4: the path of the most recently selected component is displayed, and its parent may be selected by clicking on the “up” button at the left. If no component is selected, you will need to select one using the navigation panel or the viewer. Once a component is selected, the combination box on the right will provide a selection of properties that may be selected for the widget. Once a property is selected, other options in the dialog may be used to tune the appearance of the widget:

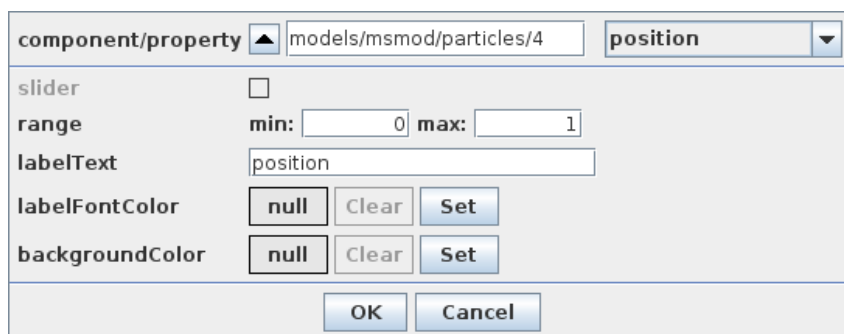


Figure 63: Widget creation dialog.

**slider**

Enabled for properties with a scalar numeric value. Setting it to true will create a widget with a slider.

**range**

Specifies the range for the slider, if one is selected.

**labelText**

The name of the widget in the panel. By default, this is the name of the property.

**labelFontColor**

Font color for the widget name. If `null`, then the default color is used.

**backgroundColor**

Background color for the widget. If `null`, then the default background is used.

**15.1.1 Composite property widgets**

A [CompositeProperty](#) is a Property which contains subproperties (see the “Composite Properties” section of the [Maspack Reference Manual](#)).

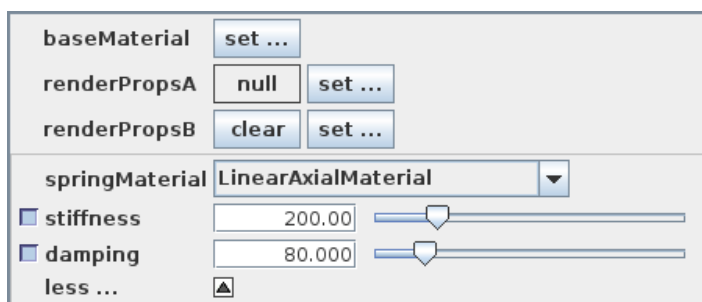


Figure 64: Composite property widgets.

If a composite property is selected, the control panel will create a *composite property widget*, the exact form of which depends on the composite property. Figure 64 shows a control panel containing four composite property widgets for the properties `baseMaterial`, `renderPropsA`, `renderPropsB`, and `springMaterial`.

The default composite property widget appears as a single set button, labeled as “set ...”, and is illustrated by the widget for `baseMaterial` in Figure 64. Clicking the set button opens another property panel that allows the subproperties of the composite property to be set.

If the composite property is allowed to have a `null` value, then this is controlled by an additional *null button* that precedes the set button. If the current property value is `null`, then the null button will display `null` (as illustrated by the widget for `renderPropsA` in Figure 64), and clicking the set button will create a new composite property instance that

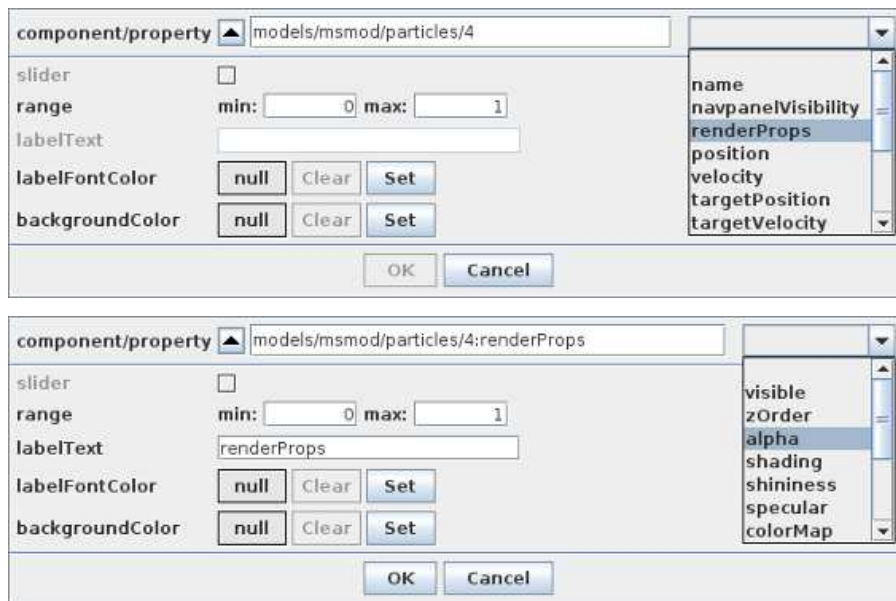


Figure 65: Selecting property `renderProps` (top), then one of its subproperties (bottom).

will replace the null value. Otherwise, if the current property value is non-null, then the null button will display clear (as illustrated by the widget for `renderPropsB` in Figure 64), and hitting it will set the current property value to null.

Finally, if the composite property has subclasses, and the types of these are exported by the static method

```
public static Class<?>[] getSubClasses();
```

in the property's class definition, then the composite property widget takes the form of a *composite property panel*, which instead of using a set button, expands all the subproperties inline within a subpanel of the control panel, together with a combination box that allows different subclass instances to be set. This is illustrated for the property `springMaterial` in Figure 64, where the composite property is an `AxialMaterial`, which has a number of subclasses, including `LinearAxialMaterial` which is currently selected in the figure. Different subclass instances may have different subproperties, and the subpanel is updated to reflect these when the subclass is changed. In order to help the control panel save space, the subpanel can also be expanded or collapsed using a more/less button at its bottom.

### 15.1.2 Widgets for subproperties

It is possible to attach widgets to the subproperties of a composite property, provided that the composite property has a non-null value.

In particular, when a non-null composite property is selected from the component/property widget, the user has the option of either

- clicking the Done button and selecting the composite property, which will create a composite property widget as described in the previous section, or
- selecting one of the composite property's subproperties.

When a non-null composite property is selected, the property's name will move over into the component field of the component/property selector, and the combination box will be cleared and reset to allow the selection of the subproperties.

For example, in Figure 65, we first select `renderProps`, which is a composite property of `models/msmod/particle/2`, and then (in the lower panel) select the sub-property `visible`. When `renderProps` is selected, its name is moved to the component panel, where it appears as

```
models/msmod/particle/2:renderProps
```

**Note:**

The ':' character is used to separate components from properties in component/property paths

## 15.2 Editing control panels

An existing control panel can also be edited. Specifically,

- Individual widgets can be moved, deleted, or have their properties set.
- Separators can be added between widgets.
- Global aspects of the control panel itself can be set.

To edit an individual widget, you first select it by left-clicking on it. This will cause it to become highlighted. You can then:


- **Move** the widget by dragging it to a different vertical location within the panel;
- **Delete** the widget by right-clicking and choosing `delete`;
- **Set properties** of the widget by right-clicking and choosing `properties`;

To add a separator, select a widget above where you want the separator, right-click and choose `add separator`.

To set global aspects of the control panel itself, right-click inside the lower-most *option pane* (the small panel at the bottom and that may, in some cases, contain option buttons such as `Close` or `Done`), and choose from the provided menu.

## 15.3 Live updating

By default, a control panel is set up to update the values of its widgets every time the viewers are rerendered. This allows one to observe property values as they evolve in time.

If you do *not* want live updating of property values, then you can disable this by clicking on the *live update icon* , which is located in the lower left of the option panel.

## 16 Component Editing

Component editing in ArtiSynth is driven by the current *selection context*: depending on what items are currently selected, different editing options will appear in the context menu. These options may allow you to add, edit, or delete components.

### 16.1 Generic edit operations

#### 16.1.1 Deletion

A set of selected components can be deleted provided that

- their parent components are editable
- none of their ancestors are selected

If the currently selected components are deletable, then a `delete` option will appear in the context menu (obtained by right-clicking in the viewer or navigation panel). Selecting this will delete the components.

If the selected components are referred to by other components, then those components will be deleted also. In this case, a dialog will be presented to the user advising of this fact and requesting confirmation.

---

Figure 66: Panel for specifying the location of a coordinate frame.

### 16.1.2 Duplication

A set of selected components may be *duplicated* provided that

- their parent components are editable
- none of their ancestors are selected
- they implement [CopyableComponent](#)

If the currently selected components are duplicatable, then a duplicate option will appear in the context menu. Selecting this will enable duplication of the components: the viewer cursor will change to cross-hairs, and the user may indicate the location for the duplicated components by left-clicking in the viewer (see Section 3.8). Duplication may be canceled by right-clicking.

Sometimes, when the components to be duplicated refer to other components, those referred components will be duplicated also. This is done when the referred components are required. For example, when duplicating an [AxialSpring](#), the two points it is attached to will also be duplicated, because AxialSprings are not permitted to exist without attached points. Such cases are indicated to the user, after the duplicate option has been selected, by expanding the current selection to include all such additional components.

### 16.1.3 Undo

Many of the operations described here are *undoable*, by choosing the Undo option from the Edit menu. The menu option will indicate the name of the operation to be undone. Hitting the ‘z’ key from within the viewer (Section 3.11) will also perform undo operations.

## 16.2 Editing panels

Many editing operations involve the creation of *editing* panels (such as Figure 68, etc.) which persist beyond the invocation of a context menu. Often, these panels are created *exclusively*, so that only one can be in existence at once. This is done by having the panel acquire a lock in the ArtiSynth editing manager. The panels are not modal, so the user can still interact with the viewer and other GUI components, but other exclusive editing panels can not be created until the current one is closed. This avoids problems associated with having two “edits” active on a the model at once.

If an exclusive editing panel is open, then other exclusive editing options will still be shown in the context menu but will be disabled.

## 16.3 Specifying position, orientation, and scaling

Sometimes, an editing panel will allow you to specify the translation and rotation associated with a [RigidTransform3d](#). Typically, this will happen when there is a need to specify the location of a spatial coordinate frame, as in the example of Figure 66.

Here, the translation and rotation correspond to the fields position and orientation. The position field is straightforward: it is just three numbers giving the position of the coordinate frame origin with respect to the base (usually world) coordinates. In Figure 66, this is the vector (1, 2, 3).

The orientation field is more complex. It corresponds to the rotation of the coordinate frame with respect to base coordinates, and is represented using an *axis-angle* format of four numbers giving the axis of the rotation, followed by the angle of rotation about this axis, in degrees. (This relies on the fact that any 3D rotation can be specified as a single rotation about a single axis.) Hence the numbers

0 1 0 60

in Figure 66 correspond to a rotation of 60 degrees about the y axis. Alternatively, the numbers

1 1 0 45

would correspond to a rotation of 45 degrees about the axis (1, 1, 0). (Note that the axis does not need to be a unit vector.) Finally, no rotation, or more precisely, the *identity* rotation, is usually represented as

1 0 0 0

i.e., *zero* rotation about the x-axis. In more general situations, one may specify not only translation and rotation but also scaling, corresponding to a more general [AffineTransform3d](#). This often occurs when reading a mesh from a file: one may wish to apply an affine transform to scale, rotate, and translate the mesh that is been read in. In such cases one will also be presented with a scale field, which accepts either a single number (to denote uniform scaling), or three numbers (to denote non-uniform scaling about the x, y, and z axes).

## 16.4 Editing MechModels

A [MechModel](#) is the central ArtiSynth component for mechanical simulation. It contains sets of mechanical components, including particles, rigid bodies, axial springs, rigid body connectors, as well as sub-models including other [MechModels](#) and finite element (FEM) models. Most of these components can be added to a [MechModel](#) graphically, as described below.

To add a component to a [MechModel](#), select the [MechModel](#) and choose the appropriate edit action shown in the context menu. A [MechModel](#) cannot be selected in the viewer, but can be selected using the navigation panel (Figure 16), or by first selecting one of its visible components in the viewer and navigating up the hierarchy to it using the up arrow of the selection display (Section 4.4).

### 16.4.1 Adding finite element models

To add a [FemModel](#) to a [MechModel](#), select the [MechModel](#) and choose “Add FemModel ...” in the context menu. This will open the editing panel shown in Figure 68, which allows the user to provide information about the model’s properties and geometry.

Default values are provided for almost all of this information; the only information that *must* be specified by the user is the model’s position (corresponding to the origin of it’s volumetric mesh). This can be done either by left-clicking in the viewer (Section 3.8), or by entering coordinates in the position field of the Location subpanel. Once a position is specified, a wireframe preview of the FEM appears in the viewer (Figure 67), showing its geometry and allowing it to be moved or rotated using an attached transformer. The user is then free to continue editing the properties and geometry information, until the model is in the desired form, at which point it can be added to the [MechModel](#) by clicking the Add button.

From top to bottom, the [FemModel](#) panel contains:

- An *instruction box* containing directions for the user.
- A *General Properties* subpanel, which allows the user to set properties for the [FemModel](#). For brevity, some of these properties are hidden and can be expanded by clicking the more... button.
- A *Location* subpanel, allowing the position and orientation to be set manually. The position corresponds to the mesh origin, while the orientation is a rotation applied to the mesh, specified in *axis-angle* format (see Section 16.3).
- A *Geometry* panel, allowing specification of the mesh geometry type and various properties specific to this type. Mesh types currently supported include Grid, Tube, Torus, Sphere, Extrusion, AnsysMesh, TetgenMesh and UCDMesh. For many of these, the associated element type can also be specified: Tet (tetrahedron), Hex (hexahedron), QuadTet (quadratic tetrahedron), QuadHex (quadratic hexadredron), and Wedge.
- An *option* panel, containing the Add button, a Clear button which resets the displayed fields to default values, and a Cancel button which closes the panel without adding a [FemModel](#).

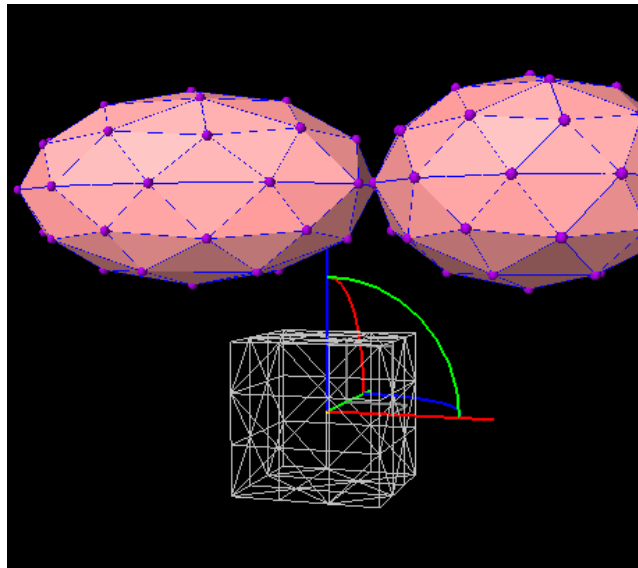


Figure 67: Wireframe preview of the FEM in the viewer.

Instructions:  
Specify position via the text field or clicking in the viewer

General Properties

name

density

more ... ☐

Location

position

orientation

Geometry

mesh type

widths

divisions

element type

Figure 68: Panel for adding finite element models.

Instructions:  
Specify position via the text field or clicking in the viewer

General Properties

name

dynamic ☒

more ...

Location

position

orientation

Geometry And Inertia

geometry type

widths

set inertia by

density

mass

rotational inertia

center of mass

Figure 69: Panel for adding RigidBodies.

### 16.4.2 Adding rigid bodies

To add a [RigidBody](#) to a MechModel, select the MechModel and choose “Add RigidBody ...” in the context menu to open an editing panel for rigid bodies, as shown in Figure 69.

As with adding FEM models, default values are provided for most information; the user must only specify the body’s position, either by left-clicking in the viewer (similar to Section 3.8), or by entering coordinates in the position field of the Location subpanel. Once a position is specified, a wireframe preview of the rigid body appears in the viewer (Figure 67), showing its geometry and allowing it to be moved or rotated using an attached transformer. The user is then free to continue editing the properties, geometry and inertia information, until the model is in the desired form, at which point it can be added to the MechModel by clicking the Add button.

From top to bottom, the Add RigidBody panel contains:

- An *instruction box* containing directions for the user.
- A *General Properties* subpanel, which allows the user to set properties for the body. For brevity, some of these properties are hidden; the panel can be expanded by clicking the more... button.
- A *Location* subpanel, allowing the position and orientation of the body’s coordinate system to be set manually. Position is specified as a three-vector, while the orientation is given as a rotation in *axis-angle* format (see Section 16.3).
- A *Geometry And Inertia* subpanel, which allows the user to specify the body’s surface mesh geometry and spatial inertia, using the same type of panel as described in Section 16.5.1.
- An *option* panel, containing the Add button, a Clear button which resets the displayed fields to default values, and a Cancel button which closes the panel without adding a rigid body.

### 16.4.3 Adding frame markers

A [FrameMarker](#) is a massless [Point](#) attached to a [RigidBody](#). It can be used for tracing motions of that body, or as an anchor point for attaching axial springs or other components.

To add one or more frame markers to the rigid bodies in a MechModel, you can select either the MechModel, or one of its rigid bodies, and then choose “Add FrameMarkers ...” in the context menu. This will open a FrameMarker editing

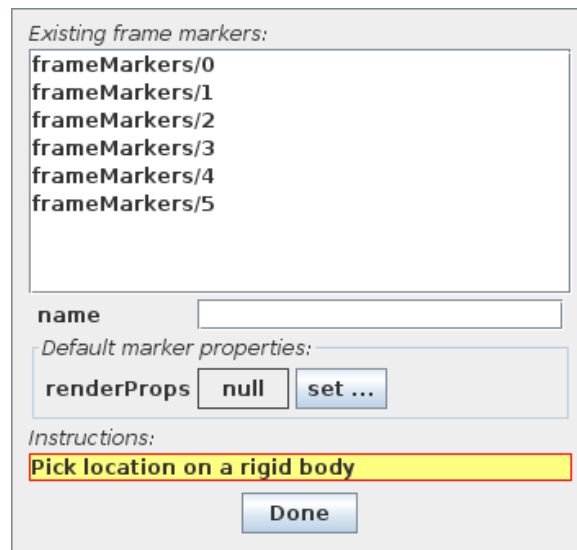


Figure 70: Panel for adding frame markers.

panel, as shown in Figure 70. While this panel is open, frame markers can be added by using the viewer and left-clicking the mouse over the surface mesh of the rigid body at the location where you want the marker to be placed (see Section 3.8). The rigid body in question must belong to the MechModel that was originally selected; no marker will be added to bodies that belong to another MechModel or a MechModel which is a submodel of the current one.

From top to bottom, the FrameMarker editing panel contains

- An *Existing frame markers* list, showing all the MechModel’s frame markers (expressed by their path names with respect to the MechModel). This list is connected to the selection manager and can be used to select one or more markers.
- A *name* field that allows a name to be specified for the marker.
- A *Default marker properties* panel, which allows the user to set properties for subsequent FrameMarkers that are added; at present, this is limited to render properties.
- An *instruction box* containing directions for the user.
- An *option* panel, which in this case contains a Done button which the user should click when finished.

#### 16.4.4 Adding particles

A **Particle** is a dynamic component, with mass, derived from **Point**. It is usually connected to other components in a model with either axial springs (Section 16.4.5) or point-to-point attachments (Section 16.4.7).

To add one or more particles to a MechModel, select the MechModel in question and choose “Add Particles ...” in the context menu. This will open a Particle editing panel, as shown in Figure 71. While this panel is open, a particle can be added by left-clicking the mouse in the viewer at the location where you want the particle to be placed, using the constrain to plane option if necessary (see Section 3.8).

From top to bottom, the Particle editing panel contains

- An *Existing particles* list, showing all the MechModel’s particles (expressed by their path names with respect to the MechModel). This list is connected to the selection manager and can be used to select one or more particles.
- A *name* field that allows a name to be specified for the particle.
- A *Default particle properties* panel, which allows the user to set properties for subsequent particles that are added. For brevity, some of these properties are hidden; the panel can be expanded by clicking the more... button.
- An *instruction box* containing directions for the user.



Figure 71: Panel for adding particles to a MechModel.

- A constrain to plane option.
- An *option* panel, which in this case contains a Done button which the user should click when finished.

#### 16.4.5 Adding axial springs and muscles

An [AxialSpring](#) is a point-to-point force effector that connects two [Points](#) and effects a force between them based on their separating distance. AxialSprings and its subclasses can be used to implement linear or nonlinear springs, as well as the subclass [Muscle](#) used to implement two-point muscles.

To add one or more axial springs to a MechModel, select the MechModel in question and choose “Add AxialSprings ...” in the context menu. This will open an AxialSpring editing panel, as shown in Figure 72. While this panel is open, axial springs can be added by selecting (using the viewer or any other selection mechanism) the two points to which the spring is attached. Points may include frame markers, particles, or FEM nodes. However, the points must be contained within the MechModel or one of its submodels.

By default, two points must be selected, in succession, for each axial spring added. Alternatively, by selecting add continuously at the bottom of the panel, a continuous sequence of springs will be created whereby the second point selected for a given spring becomes the first point for the spring following it.

From top to bottom, the AxialSpring editing panel contains

- An *Existing axial springs* list, showing all the MechModel’s springs (expressed by the path names, with respect to the MechModel, of their points). This list is connected to the selection manager and can be used to select one or more springs.
- A *name* field that allows a name to be specified for the spring.
- A *Spring type* field that allows a specific subclass of AxialSpring to be selected.
- A *Default properties* panel, which allows the user to set properties for subsequent springs that are added. The properties in question vary depending on the type selected in the Spring type field, but will include render properties and the spring material. For brevity, some properties may be hidden, in which case the panel can be expanded by clicking the more... button.
- A *progress* field displaying the path names of the points as they are selected.
- An *instruction box* containing directions for the user.
- An add continuously option as described above.

Existing axial springs:

- frameMarkers/0 - frameMarkers/2
- frameMarkers/1 - particles/red
- particles/red - frameMarkers/3

name

Spring type **AxialSpring** ▼

Default AxialSpring properties:

renderProps

material **LinearAxialMaterial** ▼

more ... ☒

Instructions:

**Select first point**

☐ add continuously

Figure 72: Panel for adding axial springs and muscles.

- An *option* panel, containing an Add/Stop button which is used to initiate or stop the addition of springs, and a Done button which the user should click when finished.

#### 16.4.6 Adding rigid body connectors

A **BodyConnector** is a component that implements constraint-based joints between either two rigid bodies, or between one rigid body and ground. The GUI currently allows two types of joints to be added: *spherical* and *revolute*.

To add one or more rigid body connectors to a MechModel, select the MechModel in question and choose “Add RigidBodyConnectors ...” in the context menu. This will open a RigidBodyConnector editing panel, as shown in Figure 73. While this panel is open, a connector can be added by selecting in succession (using the viewer or any other selection mechanism) the rigid bodies associated with it. For the case of a single rigid body connected to ground, the user clicks the Fixed button instead of selecting a second body.

After the bodies have been selected, the connector location must then be specified by left-clicking in the viewer (Section 3.8). By default, the orientation of the connector is aligned with the world axes. This can be adjusted later using the dragger fixtures (Section 5.1).

From top to bottom, the RigidBodyConnector editing panel contains

- An *Existing rigid body connectors* list, showing all the MechModel’s connectors (expressed by the path names, with respect to the MechModel, of their rigid bodies). This list is connected to the selection manager and can be used to select one or more connectors.
- A *name* field that allows a name to be specified for the connector.
- A *Connector type* field that allows a specific connector type to be selected.
- A *Default properties* panel, which allows the user to set properties for subsequent connectors that are added. The properties in question vary depending on the type selected in the Connector type field. For brevity, some properties may be hidden, in which case the panel can be expanded by clicking the more... button.
- A *progress* field displaying the path names of the rigid bodies as they are selected.

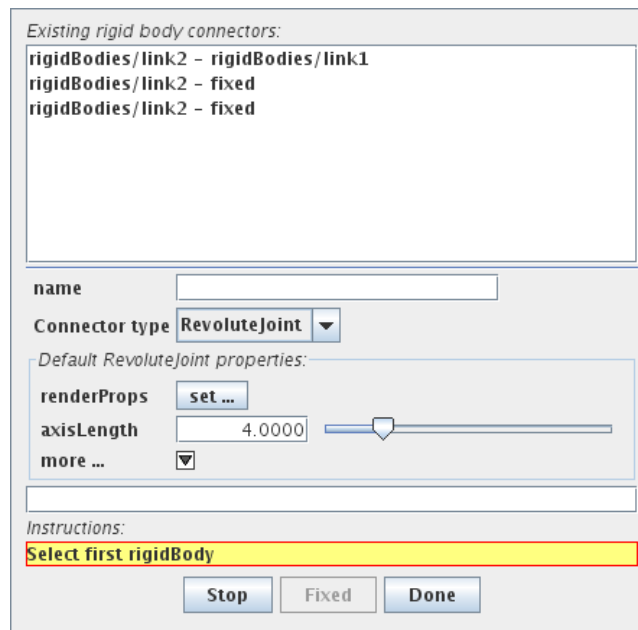


Figure 73: Panel for adding rigid body connectors.

- An *instruction box* containing directions for the user.
- An *option* panel, containing an Add/Stop button which can be used to initiate or stop the addition of connectors, a Fixed button used to indicate when a rigid body is to be connected to ground, and a Done button which the user should click when finished.

#### 16.4.7 Attaching particles to particles

Within a MechModel, two particles or FEM nodes can be attached together, resulting in what is essentially a single particle that combines the dynamics of both original particles. In particular, these *particle attachments* are a convenient way to connect FEM models to other FEM models or to particles within a MechModel.

To attach particles contained within a MechModel, select the MechModel in question and choose “Attach particles ...” in the context menu. This will open a ParticleAttachment panel, as shown in Figure 74. While this panel is open, pairs of particles can be attached by selecting in succession (using the viewer or any other selection mechanism) the two particles to be connected.

From top to bottom, the ParticleAttachment panel contains

- An *Existing attachments* list, showing all the MechModel’s attachments (expressed by the path names, with respect to the MechModel, of their particles). This list is connected to the selection manager and can be used to select one or more attachments.
- A *progress* field displaying the path names of the particles as they are selected.
- An *instruction box* containing directions for the user.
- An *option* panel, containing an Attach/Stop button which can be used to initiate or stop the attachment process, and a Done button which the user should click when finished.

#### 16.4.8 Attaching particles to rigid bodies

Within a MechModel, particles and FEM nodes can also be attached to rigid bodies. In particular, this provides a way to connect FEM models to rigid bodies within a MechModel.

To attach particles to a rigid body, select the rigid body in question and choose “Attach particles ...” in the context menu. This will open a ParticleRigidBodyAttachment panel, as shown in Figure 75. While this panel is open, particles can be

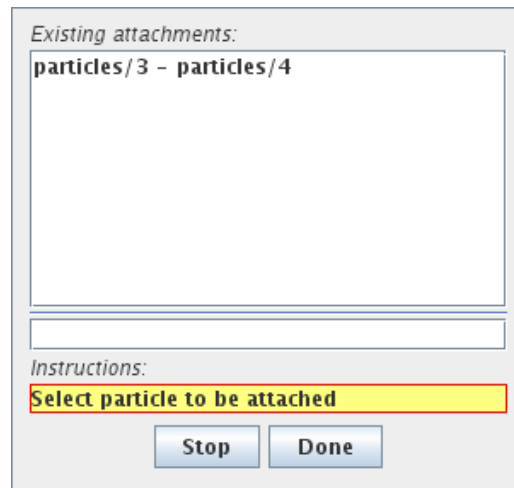


Figure 74: Panel for attaching particles together.

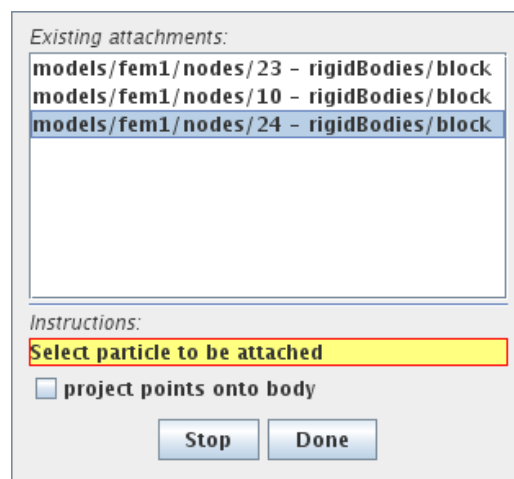


Figure 75: Panel for attaching particles to a rigid body.

attached to the body by selecting them in succession. By default, the attached particles remain where they are, so that the attachment point is determined by the current particle location relative to the rigid body's coordinates. However, this will typically not coincide with the body's surface mesh. By selecting project points onto body at the bottom of the panel, attached points will be relocated to the nearest location on the surface mesh as they are selected.

From top to bottom, the ParticleRigidBodyAttachment panel contains

- An *Existing attachments* list, showing all the MechModel's particle to rigid body attachments (expressed by the path names, with respect to the MechModel, of the particles and the bodies). This list is connected to the selection manager and can be used to select one or more attachments.
- A *progress* field displaying the path names of the particles as they are selected.
- A project points onto body field, described above.
- An *instruction box* containing directions for the user.
- An *option* panel, containing an Attach/Stop button which can be used to initiate or stop the attachment process, and a Done button which the user should click when finished.

#### 16.4.9 Collision handling

In ArtiSynth, collision detection and handling can be enabled between rigid bodies (such as [RigidBody](#)), deformable bodies (such as [FemModel3d](#)), and more generally any body that implements the interface [Collidable](#). Self intersection

	enable	friction
rigid-rigid	<input checked="" type="checkbox"/>	0.2
rigid-deformable	<input checked="" type="checkbox"/>	0.1
deformable-deformable	<input type="checkbox"/>	0
deformable-self	<input type="checkbox"/>	0

Set Cancel

Figure 76: Dialog for setting default collision behavior in a MechModel.

enabled ☒

friction 0.1

Set Cancel

Figure 77: Dialog for setting collision behavior between bodies.

is not directly supported, but is indirectly supported for compound deformable bodies that contain sub-collidable components. For example, an FEM model is a compound collidable that may contain multiple surface meshes, some of which can be made to collide with each other. For more details on collision handling, see the “Collision Handling” section of the [ArtiSynth Modeling Guide](#).

The collision response between any two pairs of bodies is determined by a [CollisionBehavior](#) component, which contains various properties controlling collision interactions. Two of these can be directly modified from the GUI:

#### enabled

whether or not collisions are enabled;

#### friction

the friction coefficient if collisions are enabled.

Collisions handling is managed by a [CollisionManager](#) component within each `MechModel`. Each `MechModel` provides four default behaviors that determine the default collision response for (a) rigid body pairs, (b) rigid-deformable body pairs, (c) deformable body pairs, and (d) deformable self-intersection. In addition to these, *override* collision behaviors can be specified for any pairs of bodies. In situations where a `MechModel` contains sub-`MechModels`, then the collision behavior for any pair of collidables is controlled by the lowest `MechModel` in the hierarchy that contains both.

There are several ways to edit collision behavior using the GUI.

- For default behaviors, the enabled and friction properties can be edited by selecting the `MechModel` and then choosing “Set default collisions ...” from the context menu. This will open the dialog shown in Figure 76, allowing the enabled and friction properties to be adjusted. For the example shown, collisions are enabled between all rigid bodies, with a friction coefficient of 0.2, and between all rigid and deformable bodies, with a coefficient of 0.1. Other collisions are disabled.
- If a user selects a particular set of rigid and/or deformable bodies, a specific collision behavior may be established among those bodies by choosing “Set collisions ...” from the context menu. That will open the dialog shown in Figure 77, allowing the enabled and friction properties for this behavior to be set.
- If a user selects a single deformable body, a specific self-intersection behavior for that body may be established by selecting “Set self collision ...”.
- More detailed collision control can be realized by selecting the `MechModel`’s collision manager in the navigation panel (Section 4.2). Choosing “Edit properties ...” or “Edit render props ...” from the right context menu then allows other properties to be set to control either the collision behavior or the rendering of collisions. A sample property panel is shown in Figure 78, and these properties are described in the “Collision Handling” section of the [ArtiSynth Modeling Guide](#).

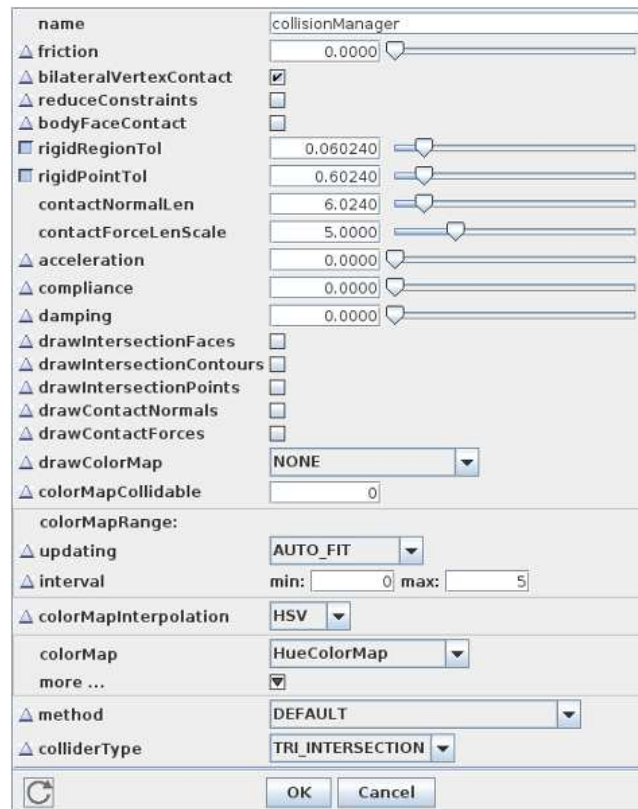


Figure 78: Property panel for the collision manager.

- For more fine-grained control, one may also use the navigation panel to select one or more of the behavior components located under the collision manager (see Figure 79). The first four of these control the default behaviors. Other behaviors, if any, are overrides that have been added either by application code, or through the GUI. Once selected, one can choose “Edit properties ...” or “Edit render props ...” from the right context menu to edit their properties. In the case of override behaviors, the context menu can also be used to remove them.
- Finally, all override behaviors in a specific `MechModel` may be removed by selecting “Remove collision overrides”. This will cause the collision behavior for all bodies to revert to default values.

## 16.5 Editing rigid bodies

The GUI provides some ability to edit rigid bodies, (type `RigidBody`), the most important of which allows the user to edit its mesh geometry and inertia (see Section 16.5.1). If a rigid body is selected, the context menu will provide the following options:

### Add FrameMarkers ...

Allows FrameMarkers to be added to the rigid body (see Section 16.4.3).

### Select markers

Causes all markers attached to the rigid body to be selected.

### Save mesh as ...

Allows the surface mesh to be saved as an Alias Wavefront `.obj` file.

### Edit geometry and inertia ...

Change the mesh geometry and/or inertia (see Section 16.5.1, below).

### Attach particles ...

Allows particles to be attached to the rigid body (see Section 16.4.8).

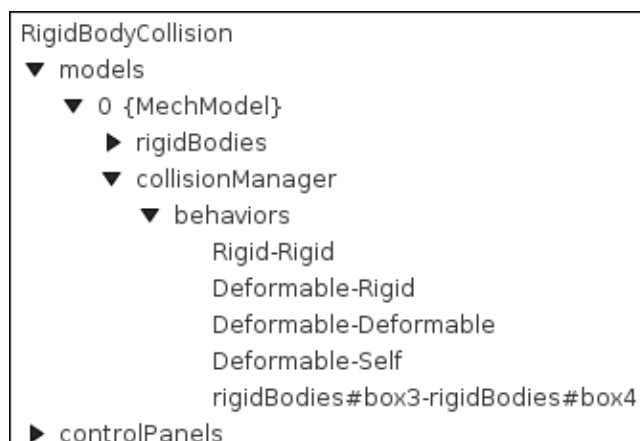


Figure 79: Expanded navigation panel showing the collision manager and individual behavior components for a MechModel.

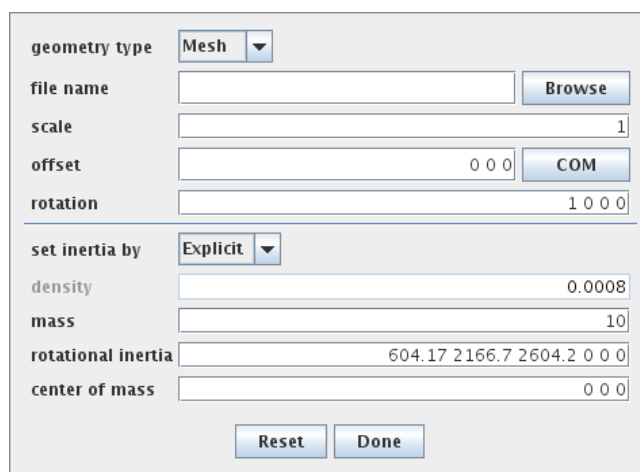


Figure 80: Panel for editing geometry and inertia.

### 16.5.1 Geometry and inertia

Choosing “Edit geometry and inertia ...” in the context menu for a rigid body opens a geometry and inertia panel, as shown in Figure 80.

The upper part of the panel allows the user to set the mesh geometry, according to a type specified by the geometry type field. Changing the geometry type changes the panel to include fields for setting parameters appropriate to the type. Currently supported types include:

#### Box

An axis-aligned box, centered with respect to body coordinates, with the x, y, and z widths set by three numbers in a widths field.

#### Sphere

A sphere, centered with respect to body coordinates, with the radius and the number of vertical mesh slices given by fields radius and slices.

**Mesh** A mesh read in from an Alias Wavefront .obj file, whose name is specified by a file name field. The read mesh can also be scaled, offset, and rotated using information provided by scale, offset, and rotation fields (see Section 16.3). The COM button causes the mesh to be offset so that its center of mass (assuming uniform density) is coincident with the origin of the body’s coordinate system (also causing the location of the center of mass to become zero).

**Note:**

At present, there appears to be a bug in the code that compute inertia from geometry, producing small errors in the center of mass calculation. That means that hitting the COM button will not cause the center of mass to become zero, but instead a small number that will converge to zero if COM is hit repeatedly.

The lower part of the panel sets the body's spatial inertia. Spatial inertia for a rigid body can be set in three ways, corresponding to the value of the body's `inertiaMethod` property:

**Density**

The spatial inertia is calculated from the density and the surface mesh geometry, with the assumption that the density is uniform.

**Mass**

The spatial inertia is calculated from the mass and the surface mesh geometry, with the assumption that the density is uniform. The density is computed by dividing the mass by the mesh volume.

**Explicit**

The spatial inertia is explicitly specified by entering values in the mass, inertia, and center of mass fields. The density is set to the average value obtained by dividing the mass by the mesh volume.

The inertia method can be set using the `set inertia by field`. Four other fields describe properties associated with the spatial inertia itself:

**density**

The mass divided by the volume

**mass**

The scale mass of the body

**rotational inertia**

The `xx`, `yy`, `zz`, `xy`, `xz`, and `yz` components of the rotational inertia tensor about the center of mass in body coordinates

**center of mass**

the position of the center of mass with respect to body coordinates.

Depending on the inertia method, the contents of these fields are either set by the user or updated automatically.

## 16.6 Editing FEM models

The GUI also provides some ability to edit FEM models (type `FemModel3d`). If an FEM model is selected, the context menu will provide the following options:

**Add FemMarkers ...**

Allows the user to add marker points to the FEM, as described in Section 16.6.1.

**Rebuild surface mesh** Rebuilds the surface mesh for the FEM. The surface mesh is computed automatically from the faces of all the elements, with inside faces being removed. Also, any elements which are fully or partly obscured by an active clipping plane are removed from the calculation, making it possible to create "partial" surface meshes that provide a cutaway view of the model.

**Save surface mesh ...**

Allows the current surface mesh to be saved to an Alias Wavefront `.obj` file.

**Save mesh as ANSYS file ...**

Allows the FEM volumetric mesh to be saved using the ANSYS file format.

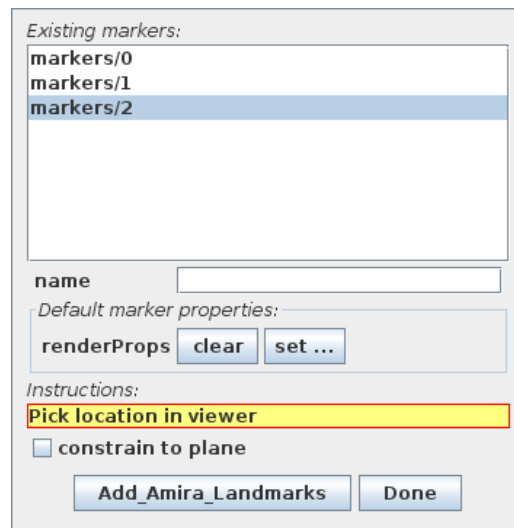


Figure 81: Panel for adding markers to an FEM model.

### 16.6.1 Adding FEM markers

A **FemMarker** is a massless **Point** attached to a specific **FemElement3d**. It can be used for tracing motions within that element, or as an anchor point for attaching axial springs or other components. It is analogous to a **FrameMarker** for FEM elements.

To add one or more markers to an FEM model, you can select the FEM model in question and then choose “Add FemMarkers ...” in the context menu. This will open a FemMarker editing panel, as shown in Figure 81. While this panel is open, FEM markers can be added by left-clicking the mouse in the viewer over the location where you want the marker placed, using the constrain to plane option if necessary (see Section 3.8). An FEM marker is then created and attached to the nearest element in the FEM. If the marker position is outside the FEM, it is projected onto the closest point on the FEM surface.

In addition, the button Add Amira Landmarks allows a user to add a set of markers based on locations in an Amira landmark file.

From top to bottom, the FemMarker editing panel contains

- An *Existing markers* list, showing all the FEM’s frame markers (expressed by their path names with respect to the FEM). This list is connected to the selection manager and can be used to select one or more markers.
- A *name* field that allows a name to be specified for the marker.
- A *Default marker properties* panel, which allows the user to set properties for subsequent markers that are added; at present, this is limited to render properties.
- An *instruction box* containing directions for the user.
- A constrain to plane option.
- An *option* panel, containing an Add Amira Landmarks button, described above, and a Done button which the user should click when finished.

### 16.6.2 Adding muscle bundles

**FemMuscleModel** is a subclass of **FemModel3d** that supports muscle activation. A **FemMuscleModel** may contain muscle bundles (type **MuscleBundle**), each of which is composed of *fibres* and *elements*. The fibres are two-point muscles connecting nodes or markers within the FEM model, with activation provided by forces acting directly on the fibre end points. The elements are a set of references to existing elements within the FEM model, each combined with an activation direction. Each element reference within the bundle provides muscle activation behavior by superimposing a transversely isotropic material behavior on top of the underlying element’s material behavior.

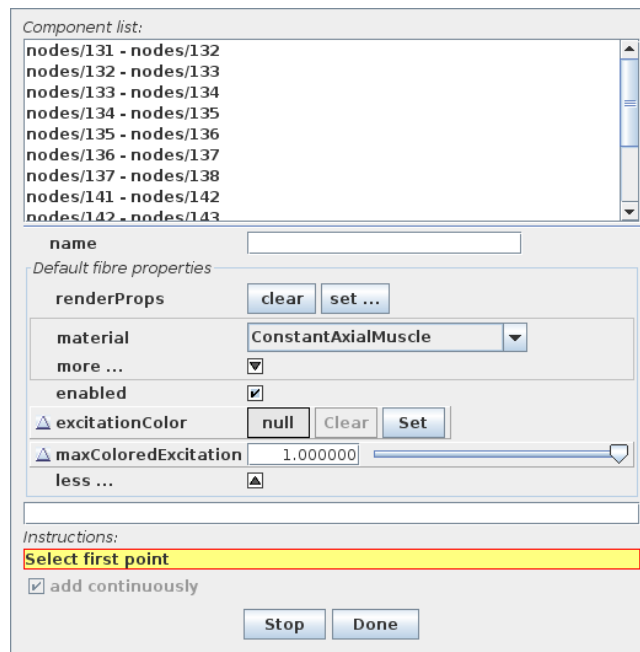


Figure 82: Panel for adding fibres to a muscle bundle.

Activation of a MuscleBundle can be effected by either the fibres *or* the elements, with the latter generally providing a superior simulation result. By default, the fibres are inactive, and are used simply to provide a good visual indication of the activation directions within the model, and a way to automatically compute the referenced elements and their directions (as described below). To make the fibres active, set the bundle's `fibresActive` property to `true`. Conversely, to make the elements inactive, set the bundle's `muscleMaterial` property to `InactiveMuscle`.

To add a MuscleBundle to a FemMuscleModel, select the model and then choose "Add MuscleBundle ..." from the context menu. This will immediately add a MuscleBundle to the model, and then open a MuscleFibre editing panel (see Section 16.7.1) to allow the user to add fibres to the model. The panel also contains two extra fields at the top: bundle name, allowing a name to be specified for the bundle, and bundle `renderProps`, allowing its render properties to be adjusted. At present, the panel does not contain a Cancel option. To remove the MuscleBundle, either select and delete it, or choose "Undo add MuscleBundle" from the Edit menu.

## 16.7 Editing muscle bundles

Existing muscle bundles can be editing to add or remove fibres or element references.

### 16.7.1 Adding fibres

Fibres can be added to a MuscleBundle by selecting the bundle and then choosing "Edit fibres ..." from the context menu. This will open a MuscleFibre editing panel as shown in Figure 82.

The operation of this panel is essentially identical to the AxialSpring editing panel described in Section 16.4.5: fibres are added by successively selecting the points (which in this case must be FEM nodes or markers) which serve as the endpoints for the fibres in question.

The only difference from the AxialSpring panel is that the spring type is assumed to be a Muscle and there is no option to change this.

### 16.7.2 Adding element references

Elements can be added to a MuscleBundle by selecting the bundle and then choosing "Edit elements ..." from the context menu. This will open a MuscleElement editing panel as shown in Figure 83.

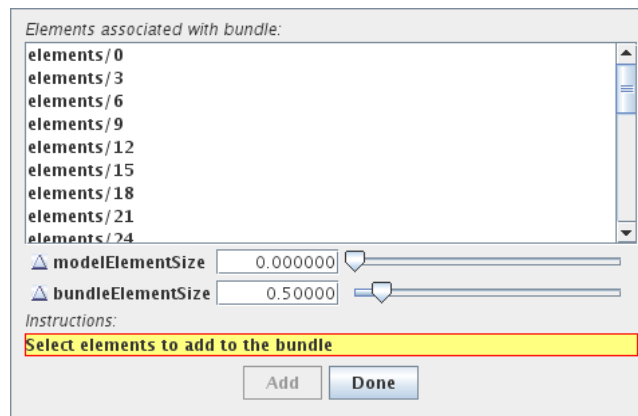


Figure 83: Panel for adding elements to a muscle bundle.

The operation of this panel is quite simple: one selects the elements that one wishes to add, and then clicks on the Add button to add them to the bundle. Elements which are already contained in the bundle will be excluded. Viewer-based element selection is described in more detail below.

From top to bottom, the MuscleElement editing panel contains

- A list of element references already associated with the bundle (expressed by the elements' path names with respect to the FEM muscle model). To remove element references from the bundle, one may select them in this list and then choose "Delete" from the context menu. It should be noted that this deletes the *references* for the elements within the bundle, and not the elements themselves from the FEM model.
- Fields modelElementSize and bundleElementSize which control the size of the element widgets which are rendered for both the FEM muscle model and the bundle, as described below.
- An *instruction box* containing directions for the user.
- An *option* panel, containing an Add button which adds selected elements to the bundle, and a Done button which the user should click when finished.

Element selection is often done by clicking on an *element widget* in the viewer. An element widget is a simplified solid rendering of an element's shape, with a size that varies from 0 to 1, with 0 being invisible and 1 being the full size of the element. Element widgets can be rendered for all the elements in an FEM model, with a size controlled by the model's elementWidgetSize property. In addition, separate widgets can be rendered for the all the elements referenced by a muscle bundle, with a size controlled by the bundle's elementWidgetSize property. In order to be able to see and select both the referenced elements in a bundle, and the other elements in the FEM model, one should set elementWidgetSize for the bundle and the model to values greater than zero, with the former larger than the latter. Figure 84 shows a simple example where referenced elements in a bundle are rendered using a widget size of 0.6, while the model elements themselves are rendered using a widget size of 0.5.

To facilitate element selection and visualization, the MuscleElement panel temporarily sets elementWidgetSize to 0.6 for the bundle and 0.5 for the FEM model. These values can then be adjusted as needed.

### 16.7.3 Automatically setting elements and directions

Since manually selecting elements and specifying their directions for a muscle bundle can be quite tedious, a number of methods exist to help do this automatically, using the easier-to-visualize information supplied by the muscle fibres. From the MuscleBundle context menu, one may select:

#### Compute element directions

Automatically computes directions for all referenced elements, using a Delaunay-based interpolation of the directions of the fibres which are closest to them.

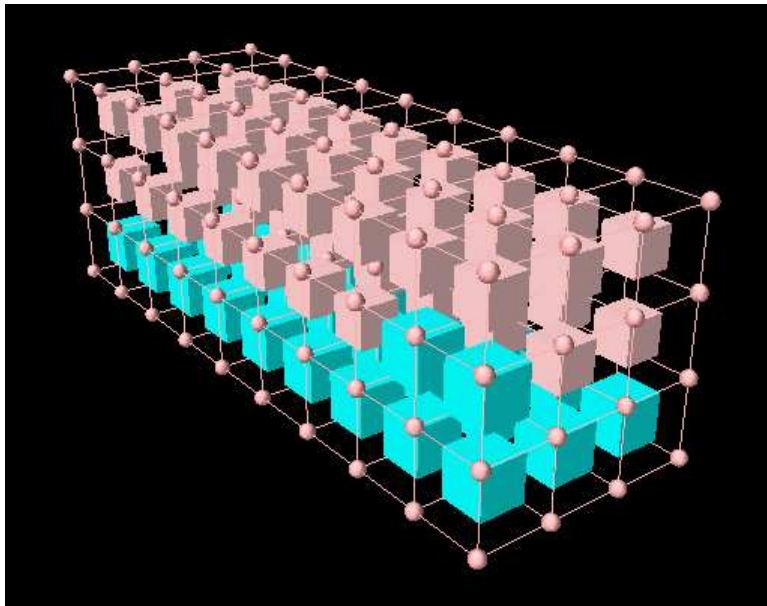


Figure 84: Element widgets rendered for both an FEM model (pink) and a muscle bundle (cyan).

#### Add elements neat fibres ...

Automatically adds to the set of referenced elements all elements whose centers are within a prescribed distance of one or more of the fibres.

#### Delete elements

Deletes all the element references for the bundle.

### 16.7.4 Removing fibres and element references

To remove specific fibres or element references, simply select them (using any of the selection mechanisms), and the choose "Delete" from the context menu.

## 16.8 Editing muscle exciters

A [MuscleExciter](#) is a component that allows muscle excitation signals to be distributed to a set of target [Excitation-Components](#). Excitation components include anything that can receive a muscle excitation, including point-to-point muscles, muscle bundles, and other muscle exciters. The purpose of a muscle exciter is to facilitate grouping so that one excitation signal can drive a number of underlying components. They can be optionally added to both MechModels and FemMuscleModels, where they are stored in a component list called `exciters`.

The GUI provides the ability to edit the targets associated with a given exciter. To do this, select the exciter in question, and then choose "Edit targets ..." in the context menu. This will open an ExcitationTarget panel, as shown in Figure 85.

To add a new excitation target, select the desired excitation component (using any of the selection mechanisms), and it will be added to the list of existing targets. Each target is also associated with a gain, by which the excitation signal is multiplied as it is passed on to the target. Gains can be edited using the numeric field in the list of targets. Finally, to remove a target, simply select it in the list of targets, and choose "remove targets" from the context menu.

From top to bottom, the ExcitationTarget panel contains

- An *Existing targets* list, showing all the current targets, allowing them to be selected for removal or their gains to be edited.
- An *instruction box* containing directions for the user.
- An *option* panel, containing an Add/Stop button which can be used to initiate or stop the adding of targets, and a Done button which the user should click when finished.

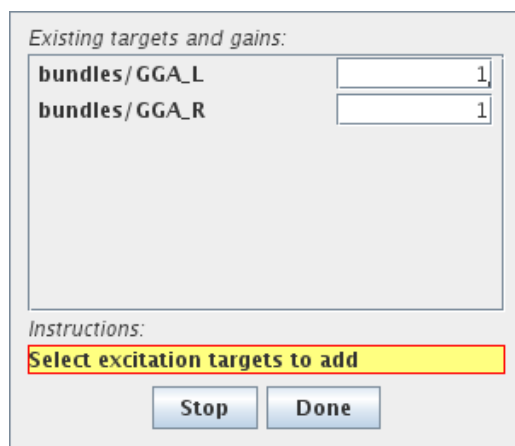


Figure 85: Panel for editing the targets of a muscle exciter.

## 16.9 Editing root models

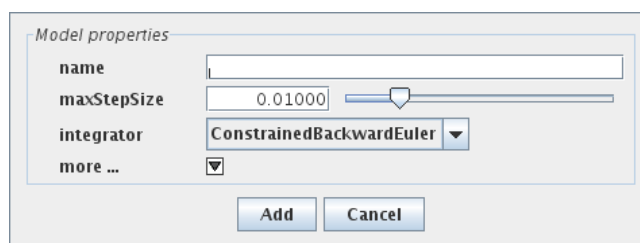


Figure 86: Panel for adding a MechModel to a RootModel.

Some very limited graphical editing is available for RootModels. It is possible to add a MechModel to the RootModel, by selecting the RootModel and then choosing "Add MechModel ..." in the context menu. This brings up a MechModel editing panel as shown in Figure 86.

The panel is quite simple: you edit the MechModel properties to the appropriate settings, click the Add button, and a new MechModel is added. However, this is of limited use, since normally only one MechModel is placed directly under the RootModel, as multiple MechModels cannot be advanced using the same integrator and must therefore be completely decoupled.